32nd International Conference on Concurrency Theory

CONCUR 2021, August 24–27, 2021, Virtual Conference

Edited by

Serge Haddad
Daniele Varacca

LIPICS

Editors

Serge Haddad
ENS Paris-Saclay, France
haddad@lsv.ens-cachan.fr

Daniele Varacca
University Paris-Est Créteil, France
daniele.varacca@u-pec.fr

ACM Classification 2012
Theory of computation → Concurrency

ISBN 978-3-95977-203-7

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-203-7.

Publication date
August, 2021

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

Digital Object Identifier: 10.4230/LIPIcs.CONCUR.2021.0

ISBN 978-3-95977-203-7 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Büchi Automata

Probabilistic Models

Games (1)

Nondeterministic and Probabilistic Models

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Distributed Systems

Language Theory

Semantics

Temporal Logics and Model Checking

Games (2)

Dynamical Systems

Equivalences

Session Types

Preface

This proceedings volume contains peer-reviewed contributions accepted at the 32st International Conference on Concurrency Theory (CONCUR), 2021.

The CONCUR conference series brings together researchers, developers, and students in order to advance the theory of concurrency, and promote its applications. Amid the COVID-19 situation, CONCUR 2021 could not take place at University Paris-Est Créteil (Créteil, France), as it was originally planned. Instead it was organized as a virtual conference, as part of the umbrella conference QONFEST 2021. In addition to CONCUR 2021, the QONFEST 2021 comprised also the 26th International Conference on Formal Methods for Industrial Critical Systems (FMICS) 2021, the 19th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS) 2021 and the 18th International Conference on Quantitative Evaluation of SysTems (QEST) 2021, alongside with several workshops and tutorials.

Despite the COVID-19 crisis, we have received a high number of submissions. Out of 96 submissions, we have accepted 35 papers for presentation at CONCUR 2021. Given great quality of many submissions, the acceptance bar was quite high. The quality criteria for acceptance were very strict and we thank our program committee and external reviewers for their excellent job in reviewing the CONCUR 2021 submissions. We are especially very grateful to all our reviewers for their efforts in providing high-quality and timely reviews and conducting active discussions on each submission at CONCUR 2021.

We thank the authors of our proceeding's papers for repaying the efforts of our reviewers and submitting their revised works to the CONCUR 2021 proceedings. We hope that details of the papers included in the present proceedings will bring lively discussions within the virtual conference platform of CONCUR 2021, initiating new research directions and collaboration within the CONCUR scientific community and beyond.

We are delighted to have had Senior Researcher Patricia Bouyer-Decitre (CNRS and LMF, France) and Prof. Davide Sangiorgi (University of Bologna, Italy) as our invited speakers. The invited talk of Prof. Boudewijn Haverkort (Tilburg University, Netherlands) was shared with QEST 2021.

Starting in 2020, a CONCUR Test-of-Time(ToT) Award has been established by the CONCUR conference and the IFIP 1.8 Working Group on Concurrency Theory. The purpose of this award is to recognize important achievements in Concurrency Theory that were published at CONCUR conferences and have stood the test of time.

For the 2021 editions, two periods are considered. Two awards were given to papers published in CONCUR between 1994 and 1997, and two more were given to papers published between 1996 and 1999. The award winners for the CONCUR ToT Awards 2021 have been selected by a jury composed of Rob van Glabbeek (chair), Luca de Alfaro, Nathalie Bertrand, Catuscia Palamidessi, and Nobuko Yoshida. The results and winners of the CONCUR ToT Award 2021 selection process are described in the invited contribution by Rob van Glabbeek in these proceedings.

We finally would like to thank the University of Paris Est, Créteil for its generous sponsorship for running CONCUR 2021. We gratefully acknowledge the sponsorship of Nomadic Labs, in particular for the best Paper Award. We also thank INRIA and the LACL laboratory for their support. We finally thank the EasyChair conference management system for assisting us in the reviewing and organization process of CONCUR 2021 together with QONFEST 2021.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

As usual, the CONCUR 2021 proceedings are open access thanks to the LIPIcs series. We thank the authors of CONCUR 2021 papers, the CONCUR 2021 participants, as well as the organizers, chaired by Benoît Barbot, and the student volunteers, for making CONCUR 2021 a sucessful virtual event.

Serge Haddad and Daniele Varacca
CONCUR 2021 PC Chairs

List of Authors

Alessandro Abate (22)
Department Computer Science, University of Oxford, UK

Michal Ajdarów (30)
Masaryk University, Brno, Czech Republic

Christel Baier (7, 28)
Technische Universität Dresden, Germany

Patrick Baillot (34)
Univ Lyon, CNRS, ENS de Lyon, Universite Claude-Bernard Lyon 1, LIP, F-69342, France

Mrudula Balachander (9)
Université libre de Bruxelles, Brussels, Belgium

A. R. Balasubramanian (17)
Technische Universität München, Germany

Nathalie Bertrand (1, 7, 15)
Université Rennes, Inria, CNRS, IRISA, France

Raven Beutner (24)
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Benedikt Bollig (14)
Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, France

Michele Boreale (19)
University of Florence, Italy

Patricia Bouyer (26)
Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, 91190, Gif-sur-Yvette, France

Léonard Brice (8)
LIGM, Univ. Gustave Eiffel, CNRS, F-77454 Marne-la-Vallée, France

Florian Bruse (23)
School of Electrical Engineering and Computer Science, Universität Kassel, Germany

Véronique Bruyère (27)
Université de Mons (UMONS), Mons, Belgium

Alessandro Cimatti (22)
Fondazione Bruno Kessler, Trento, Italy

Ornela Dardha (36)
University of Glasgow, UK

Luca de Alfaro (1)
UC Santa Cruz, CA, USA

Erik de Vink (31)
Eindhoven University of Technology, The Netherlands

Cinzia Di Giusto (14)
Université Côte d'Azur, CNRS, I3S, France

Kyveli Doveri (3)
IMDEA Software Institute, Madrid, Spain; Universidad Politécnica de Madrid, Spain

Javier Esparza (5)
Technische Universität München, Germany

Bernd Finkbeiner (24)
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Alain Finkel (14)
Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, France; Institut Universitaire de France, Paris, France

Simon Foster (20)
University of York, UK

Simon Fowler (36)
University of Glasgow, UK

Florian Funke (28)
Technische Universität Dresden, Germany

Pierre Ganty (3)
IMDEA Software Institute, Madrid, Spain

Alexis Ghyselen (34)
Univ Lyon, CNRS, ENS de Lyon, Universite Claude-Bernard Lyon 1, LIP, F-69342, France

Daniele Gorla (19)
"Sapienza" University of Rome, Italy

Jan Friso Groote (31)
Eindhoven University of Technology, The Netherlands

Ji Guan (13)
State Key Lab of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing, China

Shibashis Guha (9)
Tata Institute of Fundamental Research, Mumbai, India

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Ichiro Hasuo (21)
The Graduate University for Advanced Studies (SOKENDAI), Hayama, Japan; National Institute of Informatics, Tokyo, Japan

Daniel Hausmann (4)
Gothenburg University, Göteborg, Sweden

Vojtěch Havlena (2)
Faculty of Information Technology, Brno University of Technology, Czech Republic

Chung-Kil Hur (20)
Seoul National University, South Korea

Peter Höfner (33)
School of Computing, Australian National University, Canberra, Australia; Data61, CSIRO, Sydney, Australia

Simon Jantsch (28)
Technische Universität Dresden, Germany

Ismaël Jecker (18)
Institute of Science and Technology, Klosterneuburg, Austria

Toghrul Karimov (28)
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Shin-ya Katsumata (21)
National Institute of Informatics, Tokyo, Japan

Stefan Kiefer (5, 6, 11)
University of Oxford, UK

Naoki Kobayashi (34)
The University of Tokyo, Japan

Wen Kokke (36)
The University of Edinburgh, UK

Mayuko Kori (21)
The Graduate University for Advanced Studies (SOKENDAI), Hayama, Japan; National Institute of Informatics, Tokyo, Japan

Alexander Kozachinskiy (10)
Department of Computer Science, University of Warwick, Coventry, UK

S. Krishna (29)
Department of Computer Science & Engineering IIT Bombay, India

Rucha Kulkarni (16)
University of Illinois at Urbana-Champaign, IL, USA

Antonín Kučera (30)
Masaryk University, Brno, Czech Republic

Jan Křetínský (5)
Technische Universität München, Germany

Martin Lange (23)
School of Electrical Engineering and Computer Science, Universität Kassel, Germany

Laetitia Laversa (14)
Université Côte d’Azur, CNRS, I3S, France

Engel Lefaucheux (28)
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Ondřej Lengál (2)
Faculty of Information Technology, Brno University of Technology, Czech Republic

Sam Lindley (36)
The University of Edinburgh, UK

Etienne Lozes (14)
Université Côte d’Azur, CNRS, I3S, France

Florian Luca (28)
School of Mathematics, Wits University, Johannesburg, South Africa; Research Group in Algebraic Structures & Applications, King Abdulaziz University, Thuwal, Saudi Arabia; Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

James C. A. Main (25)
UMONS - Université de Mons, Belgium

Rupak Majumdar (35)
Max Planck Institute for Software Systems, Kaiserslautern, Germany

Jan Martens (31)
Eindhoven University of Technology, The Netherlands

Umang Mathur (16)
University of Illinois at Urbana-Champaign, IL, USA

Richard Mayr (11, 12)
School of Informatics, University of Edinburgh, UK

Nicolas Mazzocchi (18)
IMDEA Software Institute, Madrid, Spain

Jingyi Mei (13)
Shanghai Key Lab of Trustworthy Computing, East China Normal University, Shanghai, China

Andrea Micheli (22)
Fondazione Bruno Kessler, Trento, Italy

Stefan Milius (4, 32)
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

J. Garrett Morris (36)
The University of Iowa, Iowa City, IA, USA

Muhammad Syifa'ul Mufid (22)
Department Computer Science, University of Oxford, UK

Madhavan Mukund (35)
Chennai Mathematical Institute, India; CNRS IRL 2000, ReLaX, Chennai, India

Eric Munday (12)
University of Edinburgh, UK

Joël Ouaknine (28)
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Youssouf Oualhadj (26)
Univ Paris Est Creteil, LACL, F-94010 Creteil, France

Catuscia Palamidessi (1)
Inria, Paris, France

Francesco Parolini (3)
Sorbonne Université, Paris, France

Andreas Pavlogiannis (16)
Aarhus University, Denmark

Jakob Piribauer (7)
Technische Universität Dresden, Germany

David Purser (28)
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Mickael Randour (25, 26)
F.R.S.-FNRS & UMONS - Université de Mons, Belgium

Francesco Ranzato (3)
University of Padova, Italy

Jean-François Raskin (8, 9, 27)
Université libre de Bruxelles, Brussels, Belgium

Ocan Sankur (7)
Université Rennes, Inria, CNRS, IRISA, France

Lutz Schröder (4, 32)
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Pavel Semukhin (6)
University of Oxford, UK

Aneesh K. Shetty (29)
Department of Computer Science & Engineering IIT Bombay, India

Mahsa Shirmohammadi (11)
Université de Paris, CNRS, IRIF, F-75013 Paris, France

Jeremy Sproston (25)
University of Torino, Italy

Felix Stutz (35)
Max Planck Institute for Software Systems, Kaiserslautern, Germany

Amrita Suresh (14)
Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, France

Clément Tamines (27)
Université de Mons (UMONS), Mons, Belgium

K. S. Thejaswini (17)
Department of Computer Science, University of Warwick, Coventry, UK

Bastien Thomas (15)
Université Rennes, Inria, CNRS, IRISA, France

Patrick Totzke (11)
Department of Computer Science, University of Liverpool, UK

Henning Urbat (4)
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Marie van den Bogaard (8)
LIGM, Univ. Gustave Eiffel, CNRS, F-77454 Marne-la-Vallée, France

Rob van Glabbeek (1, 33)
Data61, CSIRO, Sydney, Australia; UNSW, Sydney, Australia

Pierre Vandenhove (26)
F.R.S.-FNRS & UMONS - Université de Mons, Belgium; Université Paris-Saclay, CNRS, ENS Paris-Saclay, Laboratoire Méthodes Formelles, 91190, Gif-sur-Yvette, France

Weiyou Wang (33)
School of Computing, Australian National University, Canberra, Australia

Maximilian Weininger (5)
Technische Universität München, Germany

Markus A. Whiteland (28)
Max Planck Institute for Software Systems, Saarland Informatics Campus, Saarbrücken, Germany

Josef Widder (15)
Informal Systems, Wien, Austria

Cas Widdershoven (6)
University of Oxford, UK

Thorsten Wißmann (32)
Radboud University, Nijmegen, The Netherlands

Petra Wolf (18)
Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Jim Woodcock (20)
University of York, UK

James Worrell (28)
Department of Computer Science, University of Oxford, UK

Ming Xu (13)
Shanghai Key Lab of Trustworthy Computing, MoE Engineering Research Center of Software/Hardware Co-design Technology and Application, East China Normal University, Shanghai, China

Nobuko Yoshida (1)
Imperial College, London, UK

Nengkun Yu (13)
Centre for Quantum Software and Information, Faculty of Engineering and Information Technology, University of Technology, Sydney, Australia

Georg Zetzsche (29)
Max Planck Institute for Software Systems, Kaiserslautern, Germany

Damien Zufferey (35)
Max Planck Institute for Software Systems, Kaiserslautern, Germany

CONCUR Test-Of-Time Award 2021

Nathalie Bertrand
Inria, Rennes, France

Luca de Alfaro
UC Santa Cruz, CA, USA

Rob van Glabbeek
Data61, CSIRO, Sydney, Australia
UNSW, Sydney, Australia

Catuscia Palamidessi
Inria, Paris, France

Nobuko Yoshida
Imperial College, London, UK

Abstract

This short article announces the recipients of the CONCUR Test-of-Time Award 2021.

2012 ACM Subject Classification Theory of computation → Concurrency

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

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.1

Category Invited Paper

1 Introduction

In 2020, the International Conference on Concurrency Theory (CONCUR) and the IFIP Working Group 1.8 on Concurrency Theory established the CONCUR Test-of-Time Award to recognize important achievements in concurrency theory that were published at the CONCUR conference and have stood the test of time. Starting with CONCUR 2024, an award event will take place every other year, and award one or two papers presented at CONCUR in the 4-year period from 20 to 17 years earlier.

During the present transient period two such award events are combined each year, going back in time even further. At CONCUR 2020, awards were given to papers presented at CONCUR during the period 1990–1995, the very first editions of this conference.

All papers published at CONCUR between 1994 and 1999 were eligible for this second installment of the award, which is presented at the 32nd International Conference on Concurrency Theory (CONCUR 2021). The conference is held on line from Paris, France, in the period 23–27 August 2021, with Serge Haddad and Daniele Varacca as chairs of the program committee. We had the honor to serve as members of the second CONCUR Test-of-Time Award Jury, and were asked to select one or two awardees for each of the periods 1994–1997 and 1996–1999.

After having made a shortlist of candidate award recipients and having thoroughly discussed their relative merits and impact on the CONCUR research community and beyond, we selected the four articles mentioned below for the award out of an abundance of excellent candidates.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 1; pp. 1:1–1:3
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 The Award Winning Contributions

2.1 Period 1994–1997

- David Janin & Igor Walukiewicz: *On the Expressive Completeness of the Propositional mu-Calculus with Respect to Monadic Second Order Logic.* CONCUR 1996
  `https://doi.org/10.1007/3-540-61604-7_60`

  This seminal paper relates the expressive power of monadic second-order logic and the μ-calculus, showing that the bisimulation-closed formulas of monadic second-order logic are equivalent to the μ-calculus. This is a very deep and insightful result, providing a contribution of foundational nature to the field of logic and computation. The paper's insight was one of the central factors contributing to the role of the μ-calculus in model checking, where it provides the computational counterpart to logics for expressing system properties. The relation between logics and the μ-calculus, which can in great part be traced to this paper, has been an extremely fruitful one, with implications in algorithms for the verification and analysis of transition systems, probabilistic systems, timed systems, games, and more.

- Uwe Nestmann & Benjamin C. Pierce: *Decoding Choice Encodings* CONCUR 1996
  `https://doi.org/10.1007/3-540-61604-7_55`

  This paper makes major strides in the study of the expressiveness of process calculi. It shows that, in a completely distributed and asynchronous setting, input-guarded choice can be simulated by parallel composition. More precisely, the paper constructs a fully distributed and divergence-free encoding from the input-choice pi-calculus into the asynchronous pi-calculus. The correctness of this encoding is demonstrated by establishing a semantic equivalence between a process and its encoding, thereby satisfying and strengthening the common quality criterion of full abstraction. As semantic equivalence it employs the asynchronous version of coupled simulation, and illuminates the surprising versatility of this notion by showing how it avoids the introduction of divergence in the encoding. This work formalizes ideas stemming from the programming language Pict, and has been very influential in the area of expressiveness in concurrency.

2.2 Period 1996–1999

- Ahmed Bouajjani, Javier Esparza & Oded Maler: *Reachability Analysis of Pushdown Automata: Application to Model-checking* CONCUR 1997
  `https://doi.org/10.1007/3-540-63141-0_10`

  This is a breakthrough paper that opened the way for the analysis of pushdown automata via model-checking techniques. The paper proposes a general class of alternating pushdown systems and defines new model checking algorithms for these systems against both linear and branching-time properties. The basic idea is simple, yet extremely elegant: using (regular) automata as representations for sets of states of pushdown automata. The paper proceeds to show that the representation is closed with respect to Boolean operators, makes membership of states decidable, and crucially, makes the predecessor operator easily computable. The approach proposed in this paper is so neat and natural that it has become a standard reference in the field of verification of infinite-state systems.

- Rajeev Alur, Thomas A. Henzinger, Orna Kupferman & Moshe Y. Vardi: *Alternating Refinement Relations* CONCUR 1998
  https://doi.org/10.1007/BFb0055622
  This paper introduces refinement relations, based on simulation and trace containment, for games, modeled as alternating transition systems. Refinement relations had been a foundational notion in formal methods, and much more broadly, in the theory of computation. In the years leading up to this paper, it had become evident that games provided the natural model for open systems, which communicate and are reactive to their environment; this paper extends the notion of simulation and trace containment to games. While conceived in a formal-methods and verification setting, the paper turned out to have broad implications, as these game refinement relations have implications for strategies in general games, and are closely related to notions of subtyping in game theory and in dynamic typing. The extension of refinement relations to games was thus a fundamental tassel in the understanding of dynamic systems, finally being put into place.

3 Concluding Remark

Interviews with the award recipients, which give some information on the historical context that led them to develop their award-winning work and on their research philosophy, are conducted in the Process Algebra Diary, maintained by Luca Aceto at https://processalgebra.blogspot.com/.

Reducing (To) the Ranks: Efficient Rank-Based Büchi Automata Complementation

Vojtěch Havlena
Faculty of Information Technology, Brno University of Technology, Czech Republic

Ondřej Lengál
Faculty of Information Technology, Brno University of Technology, Czech Republic

Abstract

This paper provides several optimizations of the rank-based approach for complementing Büchi automata. We start with Schewe's theoretically optimal construction and develop a set of techniques for pruning its state space that are key to obtaining small complement automata in practice. In particular, the reductions (except one) have the property that they preserve (at least some) so-called *super-tight runs*, which are runs whose ranking is *as tight as possible*. Our evaluation on a large benchmark shows that the optimizations indeed significantly help the rank-based approach and that, in a large number of cases, the obtained complement is the smallest from those produced by state-of-the-art tools for Büchi complementation.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Büchi automata, rank-based complementation, super-tight runs

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.2

Related Version *Full Version*: `https://arxiv.org/abs/2010.07834`

Acknowledgements We thank reviewers of this and previously submitted versions of the paper for their useful remarks that helped us improve the quality of the paper. This work was supported by the Czech Science Foundation project 20-07487S and the FIT BUT internal project FIT-S-20-6427.

1 Introduction

Büchi automata (BA) complementation remains an intensively studied problem since 1962, when Büchi introduced the automata model over infinite words as a foundation for a decision procedure of a fragment of a second-order arithmetic [7]. Since then, efficient BA complementation became an important task from both theoretical and practical side. It is a crucial operation in some approaches for termination analysis of programs [12, 18, 9] as well as in decision procedures concerning reasoning about programs and computer systems, such as S1S [7] or the temporal logics ETL and QPTL [34].

Büchi launched a hunt for an optimal and efficient complementation technique with his doubly exponential complementation approach [7]. A couple of years later, Safra proposed a complementation via deterministic Rabin automata with an $n^{\mathcal{O}(n)}$ *upper bound* of the size of the complement. Simultaneously with finding an efficient complementation algorithm, another search for the theoretical *lower bound* was under way. Michel showed in [28] that a lower bound of the size of a complement BA is $n!$ (approx. $(0.36n)^n$). This result was further refined by Yan to $(0.76n)^n$ in [40]. From the theoretical point of view, it seemed that the problem was already solved since Safra's construction asymptotically matched the lower bound. From the practical point of view, however, a factor in the exponent has a great impact on the size of the complemented automaton (and, consequently, also affects the performance of real-world applications). This gap became a topic of many works [22, 13, 39, 19, 41]. The efforts finally led to the construction of Schewe in [33] producing complement BAs whose sizes match the lower bound modulo a $\mathcal{O}(n^2)$ polynomial factor.

Schewe's construction stores in a macrostate partial information about all runs over some word in an input BA. In order to track the information about all runs, a macrostate contains a set of states representing a single level in a run DAG of some word with a number assigned to each state representing its rank. The number of macrostates (and hence the size of the complement) is combinatorially related to the maximum rank that occurs in macrostates.

Although the construction of Schewe is worst-case optimal, it may in practice still generate a lot of states that are not necessary. In this work, we propose novel optimizations that (among others) reduce this maximum considered rank. We build on the novel notion of a *super-tight run*, i.e., a run in the complement that uses as small ranks as possible. The macrostates not occurring in some super-tight run can be safely removed from the automaton. Further, based on reasoning about super-tight runs, we are able to reduce the maximum rank within a macrostate. In particular, we reduce the maximum considered ranking using a reasoning about the deterministic support of an input automaton or by a relation based on direct simulation implying rank ordering computed *a priori* from the input automaton. The developed optimizations give, to the best of our knowledge, **the most competitive BA complementation procedure**, as witnessed by our experimental evaluation.

These optimizations require some additional computational cost, but from the perspective of BA complementation, their cost is still negligible and, as we show in our experimental evaluation, their effect on the size of the output is often profound, in many cases by one or more orders of magnitude. Rank-based complementation with our optimizations is now competitive with other approaches, in a large number of cases (21 %) obtaining a strictly smaller complement than *any other existing tool* and in the majority of cases (63 %) obtaining an automaton at least as small as the smallest automaton provided by any other tool.

2 Preliminaries

We fix a finite nonempty alphabet Σ and the first infinite ordinal $\omega = \{0, 1, \ldots\}$. For $n \in \omega$, by $[n]$ we denote the set $\{0, \ldots, n\}$. An (infinite) word α is represented as a function $\alpha \colon \omega \to \Sigma$ where the i-th symbol is denoted as α_i. We abuse notation and sometimes also represent α as an infinite sequence $\alpha = \alpha_0\alpha_1 \ldots$ The suffix $\alpha_i\alpha_{i+1} \ldots$ of α is denoted by $\alpha_{i:\omega}$. We use Σ^ω to denote the set of all infinite words over Σ. Furthermore, for a total function $f \colon X \to Y$ and a partial function $h \colon X \rightharpoonup Y$, we use $f \lhd h$ to denote the total function $g \colon X \to Y$ defined as $g(x) = h(x)$ when $h(x)$ is defined and $g(x) = f(x)$ otherwise. Moreover, we use $\mathrm{img}(f)$ to denote the *image* of f, i.e., $\mathrm{img}(f) = \{f(x) \in Y \mid x \in X\}$ and for a set $C \subseteq X$ we use $f_{|C}$ to denote the *restriction* of f to C, i.e., $f_{|C} = f \cap (C \times Y)$.

Büchi automata. A (nondeterministic) *Büchi automaton* (BA) over Σ is a quadruple $\mathcal{A} = (Q, \delta, I, F)$ where Q is a finite set of *states*, δ is a *transition function* $\delta \colon Q \times \Sigma \to 2^Q$, and $I, F \subseteq Q$ are the sets of *initial* and *accepting* states respectively. We sometimes treat δ as a set of transitions $p \xrightarrow{a} q$, for instance, we use $p \xrightarrow{a} q \in \delta$ to denote that $q \in \delta(p, a)$. Moreover, we extend δ to sets of states $P \subseteq Q$ as $\delta(P, a) = \bigcup_{p \in P} \delta(p, a)$. We use $\delta^{-1}(q, a)$ to denote the set $\{s \in Q \mid s \xrightarrow{a} q \in \delta\}$. For a set of states S we define *reachability* from S as $\mathit{reach}_\delta(S) = \mu Z.\, S \cup \bigcup_{a \in \Sigma} \delta(Z, a)$. A *run* of $\mathcal{A}$ from $q \in Q$ on an input word α is an infinite sequence $\rho \colon \omega \to Q$ that starts in q and respects δ, i.e., $\rho_0 = q$ and $\forall i \geq 0 \colon \rho_i \xrightarrow{\alpha_i} \rho_{i+1} \in \delta$. Let $\inf(\rho)$ denote the states occurring in ρ infinitely often. We say that ρ is *accepting* iff $\inf(\rho) \cap F \neq \emptyset$. A word α is accepted by $\mathcal{A}$ from a state $q \in Q$ if there is an accepting run ρ of $\mathcal{A}$ from q, i.e., $\rho_0 = q$. The set $\mathcal{L}_\mathcal{A}(q) = \{\alpha \in \Sigma^\omega \mid \mathcal{A} \text{ accepts } \alpha \text{ from } q\}$ is called

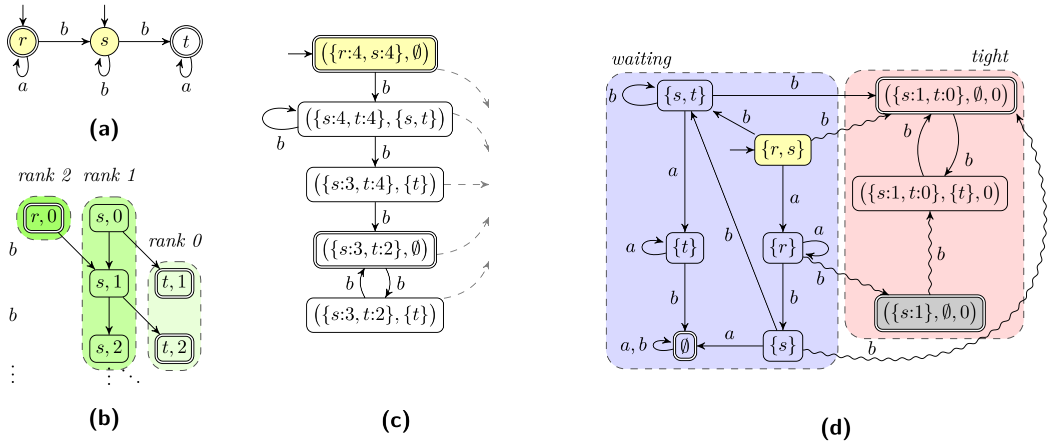


Figure 1 (a) $\mathcal{A}_{ex}$. (b) The run DAG of $\mathcal{A}_{ex}$ over b^ω. (c) A part of $\mathrm{KV}(\mathcal{A}_{ex})$. (d) $\textsc{Schewe}(\mathcal{A}_{ex})$; we highlight the *waiting* and the *tight* parts. Delay (Section 4.1) will remove the 4 wobbly transitions and macrostate $(\{s{:}1\}, \emptyset, 0)$.

the *language* of q (in $\mathcal{A}$). Given a set of states $R \subseteq Q$, we define the language of R as $\mathcal{L}_\mathcal{A}(R) = \bigcup_{q \in R} \mathcal{L}_\mathcal{A}(q)$ and the language of $\mathcal{A}$ as $\mathcal{L}(\mathcal{A}) = \mathcal{L}_\mathcal{A}(I)$. For a pair of states p and q in $\mathcal{A}$, we use $p \subseteq_\mathcal{L} q$ to denote $\mathcal{L}_\mathcal{A}(p) \subseteq \mathcal{L}_\mathcal{A}(q)$. $\mathcal{A}$ is complete iff for every state q and symbol a, it holds that $\delta(q, a) \neq \emptyset$. In this paper, we fix a BA $\mathcal{A} = (Q, \delta, I, F)$.

Simulation. The *(maximum) direct simulation* on $\mathcal{A}$ is the relation $\preceq_{di} \subseteq Q \times Q$ defined as the largest relation s.t. $p \preceq_{di} q$ implies

(i) $p \in F \Rightarrow q \in F$ and

(ii) $p \xrightarrow{a} p' \in \delta \Rightarrow \exists q' \in Q \colon q \xrightarrow{a} q' \in \delta \land p' \preceq_{di} q'$ for each $a \in \Sigma$.

Note that $\preceq_{di}$ is a preorder and $\preceq_{di} \subseteq \subseteq_\mathcal{L}$ [27].

3 Complementing Büchi Automata

In this section we first describe the basic rank-based complementation algorithm proposed by Kupferman and Vardi in [22] and then its optimization presented by Schewe in [33]. After that, we present some results related to runs with the minimal ranking. Missing proofs for this and the following section can be found in [16].

3.1 Run DAGs

In this section, we recall the terminology from [33] (which is a minor modification of the terminology from [22]), which we use heavily in the paper. We fix the definition of the *run DAG* of $\mathcal{A}$ over a word α to be a DAG (directed acyclic graph) $\mathcal{G}_\alpha = (V, E)$ of vertices V and edges E where

- $V \subseteq Q \times \omega$ s.t. $(q, i) \in V$ iff there is a run ρ of $\mathcal{A}$ from I over α with $\rho_i = q$,
- $E \subseteq V \times V$ s.t. $((q, i), (q', i')) \in E$ iff $i' = i + 1$ and $q' \in \delta(q, \alpha_i)$.

Given $\mathcal{G}_\alpha$ as above, we will write $(p, i) \in \mathcal{G}_\alpha$ to denote that $(p, i) \in V$. We call (p, i) *accepting* if p is an accepting state. $\mathcal{G}_\alpha$ is *rejecting* if it contains no path with infinitely many accepting vertices. A vertex $v \in \mathcal{G}_\alpha$ is *finite* if the set of vertices reachable from v is finite, *infinite* if it is not finite, and *endangered* if v cannot reach an accepting vertex.

We assign ranks to vertices of run DAGs as follows: Let $\mathcal{G}_\alpha^0 = \mathcal{G}_\alpha$ and $j = 0$. Repeat the following steps until the fixpoint or for at most $2n+1$ steps, where $n = |Q|$.

- Set $\mathit{rank}_\alpha(v) := j$ for all finite vertices v of $\mathcal{G}_\alpha^j$ and let $\mathcal{G}_\alpha^{j+1}$ be $\mathcal{G}_\alpha^j$ minus the vertices with the rank j.
- Set $\mathit{rank}_\alpha(v) := j+1$ for all endangered vertices v of $\mathcal{G}_\alpha^{j+1}$ and let $\mathcal{G}_\alpha^{j+2}$ be $\mathcal{G}_\alpha^{j+1}$ minus the vertices with the rank $j+1$.
- Set $j := j+2$.

For all vertices v that have not been assigned a rank yet, we assign $\mathit{rank}_\alpha(v) := \omega$. See Figure 1a for an example BA $\mathcal{A}_{ex}$ and Figure 1b for the run DAG of $\mathcal{A}_{ex}$ over b^ω.

3.2 Basic Rank-Based Complementation

The intuition in rank-based complementation algorithms is that states in the complemented automaton $\mathcal{C}$ *track* all runs of the original automaton $\mathcal{A}$ on the given word and the possible *ranks* of each of the runs. Loosely speaking, an accepting run of a complement automaton $\mathcal{C}$ on a word $\alpha \notin \mathcal{L}(\mathcal{A})$ represents the run DAG of $\mathcal{A}$ over α (in the complement, each state in a *macrostate* is assigned a *rank*)[1].

The complementation procedure works with the notion of level rankings of states of $\mathcal{A}$, originally proposed in [22, 13]. For $n = |Q|$, a *(level) ranking* is a function $f\colon Q \to [2n]$ such that $\{f(q_f) \mid q_f \in F\} \subseteq \{0, 2, \ldots, 2n\}$, i.e., f assigns even ranks to accepting states of $\mathcal{A}$. We use $\mathcal{R}$ to denote the set of all rankings and $\mathit{odd}(f)$ to denote the set of states given an odd ranking by f, i.e. $\mathit{odd}(f) = \{q \in Q \mid f(q) \text{ is odd}\}$. For a ranking f, the *rank* of f is defined as $\mathit{rank}(f) = \max\{f(q) \mid q \in Q\}$. We use $f \leq f'$ iff for every state $q \in Q$ we have $f(q) \leq f'(q)$ and $f < f'$ iff $f \leq f'$ and there is a state $p \in Q$ with $f(p) < f'(p)$.

The simplest rank-based procedure, called KV, constructs the BA $\mathrm{KV}(\mathcal{A}) = (Q', \delta', I', F')$ whose components are defined as follows [22]:

- $Q' = 2^Q \times 2^Q \times \mathcal{R}$ is a set of *macrostates* denoted as (S, O, f),
- $I' = \{I\} \times \{\emptyset\} \times \mathcal{R}$,
- $(S', O', f') \in \delta'((S, O, f), a)$ iff
  - $S' = \delta(S, a)$,
  - for every $q \in S$ and $q' \in \delta(q, a)$ it holds that $f'(q') \leq f(q)$, and
  - $O' = \begin{cases} \delta(S,a) \setminus \mathit{odd}(f') & \text{if } O = \emptyset, \\ \delta(O,a) \setminus \mathit{odd}(f') & \text{otherwise, and} \end{cases}$
- $F' = 2^Q \times \{\emptyset\} \times \mathcal{R}$.

The macrostates (S, O, f) of $\mathrm{KV}(\mathcal{A})$ are composed of three components. The S component tracks all runs of $\mathcal{A}$ over the input word in the same way as determinization of an NFA. The O component, on the other hand, tracks all runs whose rank has been even since the last cut-point (a point where $O = \emptyset$). The last component, f, assigns every state in S a rank. Note that the f component is responsible for the nondeterminism of the complement (and also for the content of the O component). A run of $\mathrm{KV}(\mathcal{A})$ is accepting if it manages to empty the O component of states occurring on the run infinitely often. We often merge S and f components and use, e.g., $(\{r{:}4, s{:}4\}, \emptyset)$ to denote the macrostate $(\{r, s\}, \emptyset, \{r \mapsto 4, s \mapsto 4\})$ (we also omit ranks of states not in S). See Figure 1c for a part of $\mathrm{KV}(\mathcal{A}_{ex})$ that starts in

[1] This is not entirely true; there may be more accepting runs of $\mathcal{C}$ over α, with ranks assigned to states of $\mathcal{A}$ that are higher than the ranks in the run DAG. There will, however, be a *minimum* run of $\mathcal{C}$ that matches the run DAG (in the terminology of Section 3.4, such a run corresponds to a *super-tight run*).

$(\{r{:}4, s{:}4\}, \emptyset)$ and keeps ranks as high as possible (the whole automaton is prohibitively large to be shown here – the implementation of KV in GOAL [37] outputs a BA with 98 states). Note that in order to accept the word b^ω, the accepting run needs to nondeterministically decrease the rank of the successor of s (the transition $(\{s{:}4, t{:}4\}, \{s, t\}) \xrightarrow{b} (\{s{:}3, t{:}4\}, \{t\})$).

In the worst case, KV constructs a BA with approximately $(6n)^n$ states [22].

3.3 Optimal Rank-Based Complementation

Friedgut, Kupferman, and Vardi observed in [13] that the KV construction generates macrostates with many rankings that are not strictly necessary in the loop part of the lasso for an accepting run on a word. Their optimization is based on composing the complement automaton from two parts: the first part (called by us the *waiting* part) just tracks all runs of $\mathcal{A}$ over the input word (in a similar manner as in a determinized NFA) and the second part (the *tight* part) in addition tracks the rank of each run in a similar manner as the KV construction, with the difference that the rankings are *tight*. For a set of states $S \subseteq Q$, we call f to be *S-tight* if

(i) it has an odd rank r,
(ii) $\{f(s) \mid s \in S\} \supseteq \{1, 3, \ldots, r\}$, and
(iii) $\{f(q) \mid q \notin S\} = \{0\}$.

A ranking is *tight* if it is Q-tight; we use $\mathcal{T}$ to denote the set of all tight rankings.

An optimal algorithm whose space complexity matches the theoretical lower bound $\mathcal{O}((0.76n)^n)$ was given by Schewe in [33, Section 3.1]. We denote this algorithm as SCHEWE. Apart from the optimization from [13], in SCHEWE, macrostates of the tight part contain one additional component, i.e., a macrostate has the form (S, O, f, i), where the last component $i \in \{0, 2, \ldots, 2n-2\}$, for $n = |Q|$, denotes the rank of states that are in O. Then, at a cut-point (when O is being reset), O is not filled with all states having an even rank, but only those whose rank is i (at every cut-point, i changes to $i + 2$ modulo the rank of f).

Formally, $\textsc{Schewe}(\mathcal{A}) = (Q', \delta', I', F')$ is constructed as follows:

- $Q' = Q_1 \cup Q_2$ where
  - $Q_1 = 2^Q$ and
  - $Q_2 = \{(S, O, f, i) \in 2^Q \times 2^Q \times \mathcal{T} \times \{0, 2, \ldots, 2n-2\} \mid f \text{ is } S\text{-tight}, O \subseteq S \cap f^{-1}(i)\}$,
- $I' = \{I\}$,
- $\delta' = \delta_1 \cup \delta_2 \cup \delta_3$ where
  - $\delta_1 : Q_1 \times \Sigma \to 2^{Q_1}$ such that $\delta_1(S, a) = \{\delta(S, a)\}$,
  - $\delta_2 : Q_1 \times \Sigma \to 2^{Q_2}$ such that $\delta_2(S, a) = \{(S', \emptyset, f, 0) \mid S' = \delta(S, a), f \text{ is } S'\text{-tight}\}$, and
  - $\delta_3 : Q_2 \times \Sigma \to 2^{Q_2}$ such that $(S', O', f', i') \in \delta_3((S, O, f, i), a)$ iff
    - ∗ $S' = \delta(S, a)$,
    - ∗ for every $q \in S$ and $q' \in \delta(q, a)$ it holds that $f'(q') \leq f(q)$,
    - ∗ $\mathit{rank}(f) = \mathit{rank}(f')$,
    - ∗ and ∘ $i' = (i + 2) \mod (\mathit{rank}(f') + 1)$ and $O' = f'^{-1}(i')$ if $O = \emptyset$ or
      ∘ $i' = i$ and $O' = \delta(O, a) \cap f'^{-1}(i)$ if $O \neq \emptyset$, and
- $F' = \{\emptyset\} \cup ((2^Q \times \{\emptyset\} \times \mathcal{T} \times \omega) \cap Q_2)$.

We call the part of $\textsc{Schewe}(\mathcal{A})$ with the states in Q_1 the *waiting* part and the part with the states in Q_2 the *tight* part (an accepting run in $\textsc{Schewe}(\mathcal{A})$ simulates the run DAG of $\mathcal{A}$ over a word w by *waiting* in Q_1 until it can generate *tight rankings* only; then it moves to Q_2). See Figure 1d for $\textsc{Schewe}(\mathcal{A}_{ex})$. Note that in order to accept the word b^ω, the accepting run needs to nondeterministically move from the waiting to the tight part.

▶ **Theorem 1** ([33, Corollary 3.3]). *Let $\mathcal{B} = \textsc{Schewe}(\mathcal{A})$. Then $\mathcal{L}(\mathcal{B}) = \overline{\mathcal{L}(\mathcal{A})}$.*

In the following, we assume that $\textsc{Schewe}(\mathcal{A})$ contains only the states and transitions reachable from I'. We use $\textsc{Schewe}$ as the base algorithm in the rest of the paper.

3.4 Super-Tight Runs

Let $\mathcal{B} = \textsc{Schewe}(\mathcal{A})$. Each accepting run of $\mathcal{B}$ on $\alpha \in \mathcal{L}(\mathcal{B})$ is *tight*, i.e., the rankings of macrostates it traverses in Q_2 are tight (this follows from the definition of Q_2). In this section, we show that there exists a *super-tight run* of $\mathcal{B}$ on α, which is, intuitively, a run that uses as little ranks as possible. Our optimizations in Section 4 are based on preserving super-tight runs of $\mathcal{B}$.

Let $\rho = S_0 \dots S_m(S_{m+1}, O_{m+1}, f_{m+1}, i_{m+1})(S_{m+2}, O_{m+2}, f_{m+2}, i_{m+2}) \dots$ be an accepting run of $\mathcal{B}$ over a word $\alpha \in \Sigma^\omega$. Given a macrostate (S_k, O_k, f_k, i_k) for $k > m$, we define its *rank* as $\mathit{rank}((S_k, O_k, f_k, i_k)) = \mathit{rank}(f_k)$. Further, we define the *rank of the run* ρ as $\mathit{rank}(\rho) = \min\{\mathit{rank}((S_k, O_k, f_k, i_k)) \mid k > m\}$. Let $\mathcal{G}_\alpha$ be the run DAG of $\mathcal{A}$ over α and rank_α be the ranking of vertices in $\mathcal{G}_\alpha$. We say that the run ρ is *super-tight* if for all $k > m$ and all $q \in S_k$, it holds that $f_k(q) = \mathit{rank}_\alpha(q, k)$. Intuitively, super-tight runs correspond to runs whose ranking faithfully copies the ranks assigned in $\mathcal{G}_\alpha$ (from some position m corresponding to the transition from the waiting to the tight part of $\mathcal{B}$).

▶ **Lemma 2.** *Let $\alpha \in \mathcal{L}(\mathcal{B})$. Then there is a super-tight accepting run ρ of $\mathcal{B}$ on α.*

Let $\rho = S_0 \dots S_m(S_{m+1}, O_{m+1}, f_{m+1}, i_{m+1})(S_{m+2}, O_{m+2}, f_{m+2}, i_{m+2}) \dots$ be a run and consider a macrostate (S_k, O_k, f_k, i_k) for $k > m$. We call a set $C_k \subseteq S_k$ a *tight core of a ranking* f_k if $f_k(C_k) = \{1, 3, \dots, \mathit{rank}(f_k)\}$ and $f_k|_{C_k}$ is injective (i.e., every state in the tight core has a unique odd rank). Moreover, C_k is a *tight core of a macrostate* (S_k, O_k, f_k, i_k) if it is a tight core of f_k. We say that an infinite sequence $\tau = C_{m+1}C_{m+2} \dots$ is a *trunk* of run ρ if for all $k > m$ it holds that C_k is a tight core of $\rho(k)$ and there is a bijection $\theta : C_k \to C_{k+1}$ s.t. if $\theta(q_k) = q_{k+1}$ then $q_{k+1} \in \delta(q_k, \alpha_k)$. We will, in particular, be interested in trunks of super-tight runs. In these runs, a trunk (there can be several) *represents runs of $\mathcal{A}$ that keep the super-tight ranks of ρ.* The following lemma shows that every state in any tight core in a trunk of such a run has at least one successor with the same rank.

▶ **Lemma 3.** *Let $\rho = S_0 \dots S_m(S_{m+1}, O_{m+1}, f_{m+1}, i_{m+1}) \dots$ be an accepting super-tight run of $\mathcal{B}$ on α. Then there is a trunk $\tau = C_{m+1}C_{m+2} \dots$ of ρ and, moreover, for every $k > m$ and all states $q_k \in C_k$, it holds that there is a state $q_{k+1} \in C_{k+1}$ such that $f_k(q_k) = f_{k+1}(q_{k+1})$.*

4 Optimized Complement Construction

In this section, we introduce our optimizations of $\textsc{Schewe}$ that are key to producing small complement automata in practice.

4.1 Delaying the Transition from Waiting to Tight

Our first optimization of the construction of the complement automaton reduces the number of nondeterministic transitions between the *waiting* and the *tight* part. This optimization is inspired by the idea of *partial order reduction* in model checking [14, 38, 29]. In particular, since in each state of the waiting part, it is possible to move to the tight part, we can arbitrarily delay such a transition (but need to take it eventually) and, therefore, significantly reduce the number of transitions (and, as our experiments later show, also significantly reduce the number of reachable states in Q_2).

Algorithm 1 The Delay construction.

```
Input: A Büchi automaton $\mathcal{A} = (Q, I, \delta, F)$
Output: A Büchi automaton $\mathcal{C}$ s.t. $\mathcal{L}(\mathcal{C}) = \overline{\mathcal{L}(\mathcal{A})}$
$\mathcal{S} \leftarrow \{I\}$, $Q_1 \leftarrow \{I\}$, $\theta_2 \leftarrow \emptyset$, $(\cdot, \delta_1 \cup \delta_2 \cup \delta_3, I', F') \leftarrow$ Schewe$(\mathcal{A})$;
while $\mathcal{S} \neq \emptyset$ do
    Take a waiting-part macrostate $R \subseteq Q$ from $\mathcal{S}$;
    foreach $a \in \Sigma$ do
        if $\exists T \in \delta_1(R, a)$ s.t. $R \xrightarrow{a} T$ closes a cycle in $Q_1$ then
            $\theta_2 \leftarrow \theta_2 \cup \{R \xrightarrow{a} U \mid U \in \delta_2(R, a)\}$;
        foreach $T \in \delta_1(R, a)$ s.t. $T \notin Q_1$ do
            $\mathcal{S} \leftarrow \mathcal{S} \cup \{T\}$;
            $Q_1 \leftarrow \mathcal{Q} \cup \{T\}$;
$Q_2 \leftarrow reach_{\delta_3}(\mathrm{img}(\theta_2))$;
return $\mathcal{C} = (Q_1 \cup Q_2, \delta_1 \cup \theta_2 \cup \delta_3, I', F' \cap Q_2)$;
```

Speaking in the terms of partial order reduction, when constructing the waiting part of the complement BA, given a macrostate $S \in Q_1$ and a symbol $a \in \Sigma$, we can set $\theta_2 \subseteq \delta_2$ such that $\theta_2(S, a) := \emptyset$ if the *cycle closing condition* holds and $\theta_2(S, a) := \delta_2(S, a)$ otherwise. Informally, the *cycle closing condition* (often denoted as **C3**) holds for S and a if the successor of S over a in the waiting part does not close a cycle where the transition to the tight part would be infinitely often delayed. Practically, it means that when constructing Q_1, we need to check whether successors of a macrostate close a cycle in the so-far generated part of Q_1. We give the construction in Algorithm 1 and refer to it as Delay. Using this optimisation on the example in Figure 1d, we would remove the b-transitions from $\{r, s\}$ and $\{s\}$ to the macrostate $(\{s{:}1, t{:}0\}, \emptyset, 0)$ and also the macrostate $(\{s{:}1\}, \emptyset, 0)$ (including the transitions incident with it).

▶ **Lemma 4.** *Let $\mathcal{A}$ be a BA. Then $\mathcal{L}(\text{Delay}(\mathcal{A})) = \mathcal{L}(\text{Schewe}(\mathcal{A}))$. Moreover, for every accepting super-tight run of Schewe$(\mathcal{A})$ on α, there is an accepting super-tight run of Delay$(\mathcal{A})$ on α.*

Since Delay does not affect the rankings in the macrostates and only delays the transition from the waiting to the tight part, we can freely use it as the base algorithm instead of Schewe in all following optimizations.

4.2 Successor Rankings

Our next optimization is used to reduce the maximum considered ranking of a macrostate in the tight part of $\mathcal{B} = \text{Schewe}(\mathcal{A})$. For a given macrostate, the number of tight rankings that can occur within the macrostate rises combinatorially with the macrostate's maximum rank (in particular, the number of tight rankings for a given set of states corresponds to the Stirling number of the second kind of the maximum rank [13]). It is hence desirable to reduce the maximum considered rank as much as possible.

The idea of our optimization called SuccRank is the following. Suppose we have a macrostate (S, O, f, i) from the tight part of $\mathcal{B}$. Further, assume that the maximum number of non-accepting states in the S-component of a macrostate that is infinitely often reachable from (S, O, f, i) is $\lceil S \rceil$. Then, we know that a super-tight accepting run that goes through (S, O, f, i) will never need a rank higher than $2\lceil S \rceil - 1$ (any accepting state will be assigned an even rank, so we can omit them). Therefore, if the rank of f is higher than $2\lceil S \rceil - 1$, we can safely discard (S, O, f, i) (since there will be a super-tight accepting run that goes over (S, O', f', i') with $f' < f$). This part of the optimization is called *coarse*.

Moreover, let $q \in S$ and let $\lfloor\{q\}\rfloor$ be the smallest size of a set of states (again without accepting states) reachable from q over some (infinite) word infinitely often. Then, we know that those states will have a rank bounded by the rank of $f(q)$, so there are only (at most) $\lceil S \rceil - \lfloor\{q\}\rfloor$ states whose rank can be higher than $f(q)$. Therefore, the rank of f, which is tight, can be at most $f(q) + 2(\lceil S \rceil - \lfloor\{q\}\rfloor)$. We call this part of the optimization *fine*.

We now formalize the intuition. Let us fix a BA $\mathcal{A} = (Q, \delta, I, F)$. Then, let us consider a BA $R_{\mathcal{A}} = (2^Q, \delta_R, \emptyset, \emptyset)$, with $\delta_R = \{R \xrightarrow{a} S \mid S = \delta(R, a)\}$, which is tracking *reachability* between set of all states of $\mathcal{A}$ (we only focus on its structure and not the language). Note that $R_{\mathcal{A}}$ is deterministic and complete. Further, given $S \subseteq Q$, let us use $SCC(S) \subseteq 2^{2^Q}$ to denote the set of all *strongly connected components reachable from S in $R_{\mathcal{A}}$*. We will use *inf-reach*(S) to denote the set of states $\bigcup SCC(S)$, i.e., the set of states such that there is an infinite path in $R_{\mathcal{A}}$ starting in S that passes through a given state infinitely many times. We define the maximum and minimum sizes of macrostates reachable infinitely often from S:

$$\lceil S \rceil = \max\{|R \setminus F| : R \in \textit{inf-reach}(S)\} \qquad \text{and} \qquad \lfloor S \rfloor = \min\{|R \setminus F| : R \in \textit{inf-reach}(S)\}.$$

For a macrostate (S, O, f, i), we define $\varphi_{coarse}((S, O, f, i)) \overset{\text{def}}{\equiv} \mathit{rank}(f) \leq 2\lceil S \rceil - 1$. If (S, O, f, i) does not satisfy φ_{coarse}, we can omit it from the output of $\textsc{Schewe}(\mathcal{A})$ (as allowed by Lemma 5). See Figure 2a for an example of such a macrostates. For instance, macrostate $(\{r{:}3, t{:}1\}, \emptyset, 0)$ can be removed since its rank is 3 and $\lceil\{r, t\}\rceil = 1$, so $3 \not\leq 2\lceil\{r, t\}\rceil - 1$.

Moreover, we also define the condition

$$\varphi_{\mathit{fine}}((S, O, f, i)) \overset{\text{def}}{\equiv} \mathit{rank}(f) \leq \min\{f(q) + 2(\lceil S \rceil - \lfloor\{q\}\rfloor) \mid q \in S\}. \tag{1}$$

Again, we can omit (S, O, f, i) if it does not satisfy φ_{fine}. See Figure 2b for an example of such a macrostate. Note that the rank of $(\{r{:}1, s{:}5, t{:}3\}, \emptyset, 0)$ is 5, $\lceil\{r, s, t\}\rceil = 3$ and $\lfloor\{r\}\rfloor = 2$, $\lfloor\{s\}\rfloor = 1$, $\lfloor\{t\}\rfloor = 0$. Then, $\min\{f(r)+2(3-2), f(s)+2(3-1), f(t)+2(3-0)\} = \min\{1+2, 5+4, 3+6\} = 3$, so the macrostate does not satisfy φ_{fine} and can be removed.

We emphasize that φ_{coarse} and φ_{fine} are incomparable. For example, the macrostates removed due to φ_{coarse} in Figure 2a satisfy φ_{fine} (since, e.g., $3 \leq \min\{3{+}2(1{-}1), 1{+}2(1{-}0)\}$) and the macrostate removed due to φ_{fine} in Figure 2b satisfies φ_{coarse} (since $5 \leq 2 \cdot 3 - 1$).

Putting the conditions together, we define the predicate

$$\textsc{SuccRank}((S, O, f, i)) \overset{\text{def}}{\equiv} \varphi_{coarse}((S, O, f, i)) \wedge \varphi_{\mathit{fine}}((S, O, f, i)). \tag{2}$$

We abuse notation and use $\textsc{SuccRank}(\mathcal{A})$ to denote the output of $\textsc{Schewe}(\mathcal{A}) = (Q', \delta', I', F')$ where the states from the tight part of Q' are restricted to those that satisfy $\textsc{SuccRank}$.

▶ **Lemma 5.** *Let $\mathcal{A}$ be a BA. Then $\mathcal{L}(\textsc{SuccRank}(\mathcal{A})) = \mathcal{L}(\textsc{Schewe}(\mathcal{A}))$.*

4.3 Rank Simulation

The next optimization $\textsc{RankSim}$ is a modification of optimization $\textsc{Purge}_{di}$ from [8]. Intuitively, $\textsc{Purge}_{di}$ is based on the fact that if a state p is directly simulated by a state r, i.e., $p \preceq_{di} r$, then any macrostate (S, O, f, i) where $f(p) > f(r)$ can be safely removed (intuitively, any run from p can be simulated by a run from r, where the run from r may contain more accepting states and so needs to decrease its rank more times). $\textsc{Purge}_{di}$ is compatible with

Schewe but, unfortunately, it is incompatible with the MaxRank construction (one of our further optimizations introduced in Section 4.5) since in MaxRank, several runs are represented by one *maximal* run (w.r.t. the ranks) and removing such a run would also remove the smaller runs. We, however, change the condition and obtain a new reduction, which is incomparable with Purge$_{di}$ but compatible with MaxRank.

Consider the following relation of *odd-rank simulation* on Q defined such that $p \preceq_{ors} r$ iff

$$\forall \alpha \in \Sigma^\omega, \forall i \geq 0\colon (\mathit{rank}_\alpha(p,i) \text{ is odd} \land \mathit{rank}_\alpha(r,i) \text{ is odd}) \Rightarrow \mathit{rank}_\alpha(p,i) \leq \mathit{rank}_\alpha(r,i). \quad (3)$$

Intuitively, if $p \preceq_{ors} r$ holds, then in any super-tight run and a macrostate (S,O,f,i) in such a run, if $p,r \in S$ and both $f(p)$ and $f(r)$ are odd, then it needs to hold that $f(p) \leq f(r)$. Such a reasoning can also be applied transitively ($\preceq_{ors}$ is by itself not transitive): if, in addition, $t \in S$, the rank $f(t)$ is odd, and $r \preceq_{ors} t$, then it also needs to hold that $f(p) \leq f(t)$.

Formally, given a ranking f, let $\preceq_{ors}^f$ be a modification of $\preceq_{ors}$ defined as

$$p \preceq_{ors}^f r \overset{\text{def}}{\equiv} f(p) \text{ is odd} \land f(r) \text{ is odd} \land p \preceq_{ors} r \quad (4)$$

and $\preceq_{ors}^{fT}$ be its transitive closure. We use $\preceq_{ors}^{fT}$ to define the following condition:

$$\textsc{RankSim}((S,O,f,i)) \overset{\text{def}}{\equiv} \forall p,r \in S\colon p \preceq_{ors}^{fT} r \Rightarrow f(p) \leq f(r). \quad (5)$$

Abusing the notation, let RankSim($\mathcal{A}$) denotes the output of Schewe($\mathcal{A}$) $= (Q',\delta',I',F')$ where states from the tight part of Q' are restricted to those that satisfy RankSim.

▶ **Lemma 6.** *Let $\mathcal{A}$ be a BA. Then $\mathcal{L}(\textsc{RankSim}(\mathcal{A})) = \mathcal{L}(\textsc{Schewe}(\mathcal{A}))$.*

From the definition of $\preceq_{ors}$, it is not immediate how to compute it, since it is defined over all infinite runs of $\mathcal{A}$ over all infinite words. The computation of a rich under-approximation of $\preceq_{ors}$ will be the topic of the rest of this section. We first note that $\preceq_{di} \subseteq \preceq_{ors}$, which is a consequence of the following lemma.

▶ **Lemma 7** (Lemma 7 in [8]). *Let $p,r \in Q$ be such that $p \preceq_{di} r$ and $\mathcal{G}_\alpha = (V,E)$ be the run DAG of $\mathcal{A}$ over α. For all $i \geq 0$, $((p,i) \in V \land (r,i) \in V) \Rightarrow \mathit{rank}_\alpha(p,i) \leq \mathit{rank}_\alpha(r,i)$.*

We extend $\preceq_{di}$ into a relation $\preceq_R$, which is computed statically on $\mathcal{A}$, and then show that $\preceq_R \subseteq \preceq_{ors}$. The relation $\preceq_R$ is defined recursively as the smallest binary relation over Q s.t.

(i) $\preceq_{di} \subseteq \preceq_R$ and
(ii) for $p,r \in Q$, if $\forall a \in \Sigma : (\delta(p,a) \setminus F) \preceq_R^{\forall\forall} (\delta(r,a) \setminus F)$, then $p \preceq_R r$.

Here, $S_1 \preceq_R^{\forall\forall} S_2$ holds iff $\forall x \in S_1, \forall y \in S_2 : x \preceq_R y$. The relation $\preceq_R$ can then be computed using a standard *worklist* algorithm, starting from $\preceq_{di}$ and adding pairs of states for which condition 2 holds until a fixpoint is reached.

▶ **Lemma 8.** *We have $\preceq_R \subseteq \preceq_{ors}$.*

Putting it all together, we modify (5) by substituting $\preceq_{ors}^{fT}$ with $\preceq_R^{fT}$, which denotes the transitive closure of $\preceq_R^f$, where $\preceq_R^f$ is a relation defined (by modifying (4)) as

$$p \preceq_R^f r \overset{\text{def}}{\equiv} f(p) \text{ is odd} \land f(r) \text{ is odd} \land p \preceq_R r. \quad (6)$$

Because $\preceq_R \subseteq \preceq_{ors}$, Lemma 6 still holds. We denote the modification of RankSim that uses $\preceq_R^{fT}$ instead of $\preceq_{ors}^{fT}$ as RankSim′.

▶ **Example 9.** Consider the BA $\mathcal{A}$ (top) and the part of Schewe($\mathcal{A}$) (bottom) in Figure 2c. Note that $r_2 \preceq_{di} q_2$ and $q_2 \preceq_{di} r_2$ so $r_2 \preceq_R q_2$ and $q_2 \preceq_R r_2$. From the definition of $\preceq_R$, we can deduce that $r_1 \preceq_R q_1$ (since $\{r_2\} \preceq_R^{\forall\forall} \{q_2\}$) and $q_1 \preceq_R r_1$ (since $\{q_2\} \preceq_R^{\forall\forall} \{r_2\}$). Note that $q_1 \not\preceq_{di} r_1$). As a consequence and due to the odd ranks of q_1 and r_1, we can eliminate the macrostates $(\{q_1{:}1, r_1{:}3\}, \emptyset, 0)$ and $(\{q_1{:}3, r_1{:}1\}, \emptyset, 0)$.

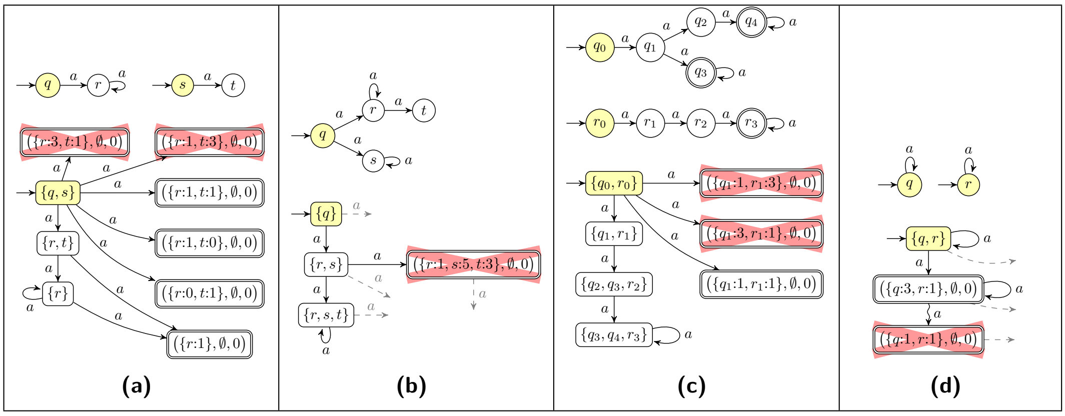


Figure 2 (a) Illustration of SuccRank reduction (φ_{coarse}), focusing on the transitions from the waiting to the tight part. (b) Illustration of SuccRank reduction (φ_{fine}), focusing on one particular macrostate. (c) Illustration of RankSim′. (d) Illustration of RankRestr.

4.4 Ranking Restriction

Another optimization, called RankRestr, restricts ranks of successors of states with an odd rank. In particular, in a super-tight run, every odd-ranked state has a successor with the same rank (this follows from the construction of the run DAG). Let $\mathcal{A}$ be a BA and $\mathcal{B} = \textsc{Schewe}(\mathcal{A}) = (Q, \delta_1 \cup \delta_2 \cup \delta_3, I, F)$. We define the following restriction on transitions:

$$\textsc{RankRestr}((S, O, f, i) \xrightarrow{a} (S', O', f', i')) \overset{\text{def}}{\equiv}$$
$$\forall q \in S : f(q) \text{ is odd } \Rightarrow (\exists q' \in \delta(q, a) : f'(q') = f(q)). \tag{7}$$

We abuse notation and use RankRestr($\mathcal{A}$) to denote $\mathcal{B}$ with transitions from δ_3 restricted to those that satisfy RankRestr. See Figure 2d for an example of a transition (and a newly unreachable macrostate) removed using RankRestr.

▶ **Lemma 10.** *Let $\mathcal{A}$ be a BA. Then $\mathcal{L}(\textsc{RankRestr}(\mathcal{A})) = \mathcal{L}(\textsc{Schewe}(\mathcal{A}))$.*

4.5 Maximum Rank Construction

Our next optimization, named MaxRank, has the biggest practical effect. We introduce it as the last one because it depends on our previous optimizations (in particular SuccRank and RankSim′). It is a modified version of Schewe's "Reduced Average Outdegree" construction [33, Section 4], named $\textsc{Schewe}_{\textsc{RedAvgOut}}$, which may omit some runs, the so-called *max-rank* runs, that are essential for our other optimizations (we discuss the particular issue later).[2]

The main idea of MaxRank is that a set of runs of $\mathcal{B} = \textsc{Schewe}(\mathcal{A})$ (including super-tight runs) that assign different ranks to non-trunk states is represented by a single, "maximal," not necessarily super-tight (but having the same rank), run in $\mathcal{C} = \textsc{MaxRank}(\mathcal{A})$. We call such runs *max-rank runs*. More concretely, when moving from the waiting to the tight part, $\mathcal{C}$ needs to correctly guess a rank that is needed on an accepting run and the first tight core

[2] We believe that this property was not originally intended by the author, since it is not addressed in the proof. As far as we can tell, the construction is correct, although the original argument of the proof in [33] needs to be corrected.

of a trunk of the run. The ranks of the rest of states are made maximal. Then, the tight part of $\mathcal{C}$ contains for each macrostate and symbol at most two successors: one via η_3 and one via η_4. Loosely speaking, the η_3-successor keeps all ranks as high as possible, while the η_4-successor decreases the rankings of all non-accepting states in O (and can therefore help emptying O, which is necessary for an accepting run).

Before we give the construction, let us first provide some needed notation. We now use $(S, O, f, i) \leq (S, O, g, i)$ to denote that $f \leq g$ and similarly for $<$ (note that non-ranking components of the macrostates need to match).

The construction is then formally defined as $\textsc{MaxRank}(\mathcal{A}) = (Q_1 \cup Q_2, \eta, I', F')$ with $\eta = \delta_1 \cup \eta_2 \cup \eta_3 \cup \eta_4$ such that $Q_1, Q_2, I', F', \delta_1$ are the same as in $\textsc{Schewe}$. Let $\mathcal{B} = \textsc{Delay}(\mathcal{A}) = (\cdot, \delta_1 \cup \theta_2 \cup \delta_3, \cdot, \cdot)$ where δ_1, θ_2, and δ_3 are defined as in $\textsc{Delay}$. We define an auxiliary transition function that uses our previous optimizations as follows:

$$\Delta^\bullet(q, a) = \{q' \mid q' \in \theta_2(q, a) \land \textsc{RankSim}'(q') \land \textsc{SuccRank}(q')\}. \tag{8}$$

(We note that q is from the waiting and q' is from the tight part of $\mathcal{B}$.) Given a macrostate (S, O, f, i) and $a \in \Sigma$, we define the maximal successor ranking $f'_{max} = \textit{max-rank}((S, O, f, i), a)$ as follows. Consider $q' \in \delta(S, a)$ and the rank $r = \min\{f(s) \mid s \in \delta^{-1}(q', a) \cap S\}$. Then

- $f'_{max}(q') := r - 1$ if r is odd and $q' \in F$ and
- $f'_{max}(q') := r$ otherwise.

Let δ_3 be the transition function of the tight part of $\textsc{Schewe}(\mathcal{A})$. We can now proceed to the definition of the missing components of $\textsc{MaxRank}(\mathcal{A})$:

- $\eta_2(S, a) := \{(S', \emptyset, f', 0) \in \Delta^\bullet(S, a) \mid (S', \emptyset, f', 0)$ is a maximal element of $\leq$ in $\Delta^\bullet(S, a)\}$.
- $\eta_3((S, O, f, i), a)$: Let $f'_{max} = \textit{max-rank}((S, O, f, i), a)$. Then, we set
  - $\eta_3((S, O, f, i), a) := \{(S', O', f'_{max}, i')\}$ when $(S', O', f'_{max}, i') \in \delta_3((S, O, f, i), a)$ (i.e., if f'_{max} is tight; note that, in general, the result of *max-rank* may not be tight) and
  - $\eta_3((S, O, f, i), a) := \emptyset$ otherwise.
- $\eta_4((S, O, f, i), a)$: Let $\eta_3((S, O, f, i), a) = \{(S', P', h', i')\}$ and let
  - $f' = h' \lhd \{u \mapsto h'(u) - 1 \mid u \in P' \setminus F\}$ and
  - $O' = P' \cap f'^{-1}(i')$.

  Then, if $i' \neq 0$, we set $\eta_4((S, O, f, i), a) := \{(S', O', f', i')\}$, else we set $\eta_4((S, O, f, i), a) := \emptyset$.

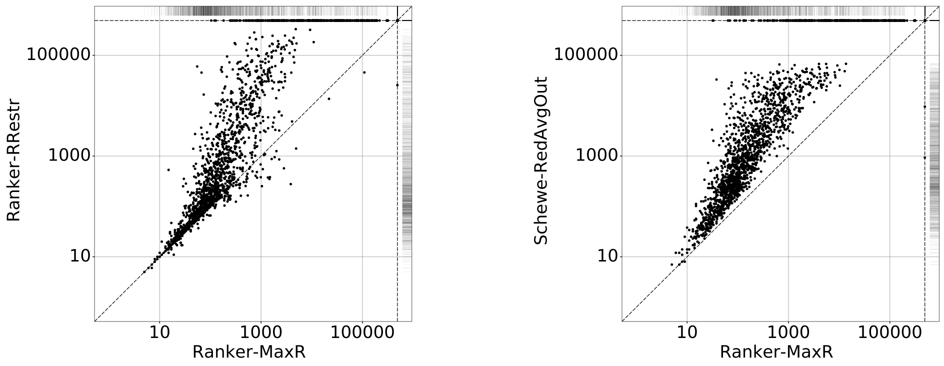
$\textsc{MaxRank}$ differs from $\textsc{Schewe}_{\textsc{RedAvgOut}}$ in the definition of η_2 and η_4. In particular, in the η_4 of $\textsc{Schewe}_{\textsc{RedAvgOut}}$ (named γ_4 therein), the condition that only non-accepting states ($u \in P' \setminus F$) decrease rank is omitted. Instead, the rank of all states in P' is decreased by one, which might create a "false ranking" (not an actual ranking since an accepting state is given an odd rank), so the target macrostate is omitted from the complement. Due to this, some max-rank runs may also be removed. Our construction preserves max-rank runs, which makes the proof of the theorem significantly more involved.

▶ **Theorem 11.** *Let $\mathcal{A}$ be a BA and $\mathcal{C} = \textsc{MaxRank}(\mathcal{A})$. Then $\mathcal{L}(\mathcal{C}) = \overline{\mathcal{L}(\mathcal{A})}$.*

Note that $\textsc{MaxRank}$ is incompatible with $\textsc{RankRestr}$ since $\textsc{RankRestr}$ optimizes the transitions in the tight part of the complement BA, which are abstracted in $\textsc{MaxRank}$.

4.6 Backing Off

Our final optimization, called $\textsc{BackOff}$, is a strategy for guessing when our optimized rank-based construction is likely (despite the optimizations) to generate too many states and when it might be helpful to give up and use a different complementation procedure instead. We evaluate this after the initial phase of $\textsc{Schewe}$, which constructs δ_2 (η_2 in



(a) RANKER$_{\text{MaxR}}$ vs RANKER$_{\text{RRestr}}$.

(b) RANKER$_{\text{MaxR}}$ vs SCHEWE$_{\text{RedAvgOut}}$.

Figure 3 Evaluation of the effectiveness of our optimizations on the generated state space (axes are logarithmic). The horizontal and vertical dashed lines represent timeouts.

MAXRANK, θ_2 in DELAY; we will just use δ_2 now), finishes. We provide a set of pairs $\{(StateSize_j, RankMax_j)\}_{j\in\mathcal{J}}$ for an index set $\mathcal{J}$ (obtained experimentally) and check (after δ_2 is constructed) that for all $(S, O, f, i) \in \text{img}(\delta_2)$ and all $j \in \mathcal{J}$ it holds that either $|S| < StateSize_j$ or $rank(f) < RankMax_j$. If for some (S, O, f, i) and j the condition does not hold, we terminate the construction and execute a different, *surrogate*, procedure.

5 Experimental Evaluation

Used tools and evaluation environment. We implemented the optimizations described in the previous sections in a tool called RANKER [17] in C++ (we tested the correctness of our implementation using SPOT's `autcross` on all BAs in our benchmark). We compared our complementation approach with other state-of-the-art tools, namely, GOAL [37] (including the FRIBOURG plugin [1]), SPOT 2.9.3 [11], SEMINATOR 2 [4], LTL2DSTAR 0.5.4 [21], and ROLL [24]. All tools were set to the mode where they output an automaton with the standard state-based Büchi acceptance condition. We note that some of the tools are aimed at complementing more general flavours of ω-automata, such as SEMINATOR 2 focusing on generalized transition-based Büchi automata. The experimental evaluation was performed on a 64-bit GNU/LINUX DEBIAN workstation with an Intel(R) Xeon(R) CPU E5-2620 running at 2.40 GHz with 32 GiB of RAM. The timeout was set to 5 minutes.

Dataset. The source of our main benchmark are the 11,000 BAs used in [36], which were randomly generated using the Tabakov-Vardi approach [35] over a two letter alphabet, starting from 15 states and with various different parameters (see [36] for more details). In preprocessing, the automata were reduced using a combination of RABIT [27] and SPOT's `autfilt` (using the `-high` simplification level) and converted to the HOA format [2]. From this set, we removed automata that are

(i) semi-deterministic,

(ii) inherently weak, or

(iii) unambiguous,

since for these kinds of automata there exist more efficient complementation procedures than for unrestricted BAs [3, 4, 5, 26]. Moreover, we removed BAs with an empty language or empty language of complement. We were left with **2,393** *hard* automata.

Table 1 Statistics for our experiments. The upper part compares different optimizations of the rank-based procedure (no postprocessing). The lower part compares our approach with other methods (with postprocessing). "BO" denotes the BackOff optimization. In the left-hand side of the table, the column "**med.**" contains the median, "**std. dev**" contains the standard deviation, and "**TO**" contains the number of timeouts (5 mins). In the right-hand side of the table, we provide the number of cases where our tool (Ranker$_{\text{MaxR}}$ without postprocessing in the upper part and with postprocessing in the lower part) was strictly better ("**wins**") or worse ("**losses**"). The "**(TO)**" column gives the number of times this was because of the timeout of the loser. Approaches implemented in GOAL are labelled with ⚽.

method	max	mean	med.	std. dev	TO	wins	(TO)	losses	(TO)
Ranker$_{\text{MaxR}}$	319 119	8 051.58	185	28 891.4	360	—	—	—	—
Ranker$_{\text{RRestr}}$	330 608	9 652.67	222	32 072.6	854	1810	(495)	109	(1)
Schewe$_{\text{RedAvgOut}}$ ⚽	67 780	5 227.3	723	10 493.8	844	2030	(486)	3	(2)
Ranker$_{\text{MaxR}}$	1 239	61.83	32	103.18	360	—	—	—	—
Ranker$_{\text{MaxR}}$+BO	1 706	73.65	33	126.8	17	—	—	—	—
Piterman ⚽	1 322	88.30	40	142.19	12	1 069	(3)	469	(351)
Safra ⚽	1 648	99.22	42	170.18	158	1 171	(117)	440	(319)
Spot	2 028	91.95	38	158.13	13	907	(6)	585	(353)
Fribourg ⚽	2 779	113.03	36	221.91	78	996	(51)	472	(333)
LTL2dstar	1 850	88.76	41	144.09	128	1 156	(99)	475	(331)
Seminator 2	1 772	98.63	33	191.56	345	1 081	(226)	428	(241)
ROLL	1 313	21.50	11	57.67	1 106	1 781	(1 041)	522	(295)

Selection of Optimizations. We use two settings of Ranker with different optimizations turned on. Since the RankRestr and MaxRank optimizations are incompatible, the main difference between the settings is which one of those two they use. The particular optimizations used in the settings are the following:

$$\begin{aligned}\textsc{Ranker}_{\text{MaxR}} &= \textsc{Delay} + \textsc{SuccRank} + \textsc{RankSim}' + \textsc{MaxRank}\\ \textsc{Ranker}_{\text{RRestr}} &= \textsc{Delay} + \textsc{SuccRank} + \textsc{RankSim}' + \textsc{RankRestr} + \textsc{Purge}_{di}\end{aligned}$$

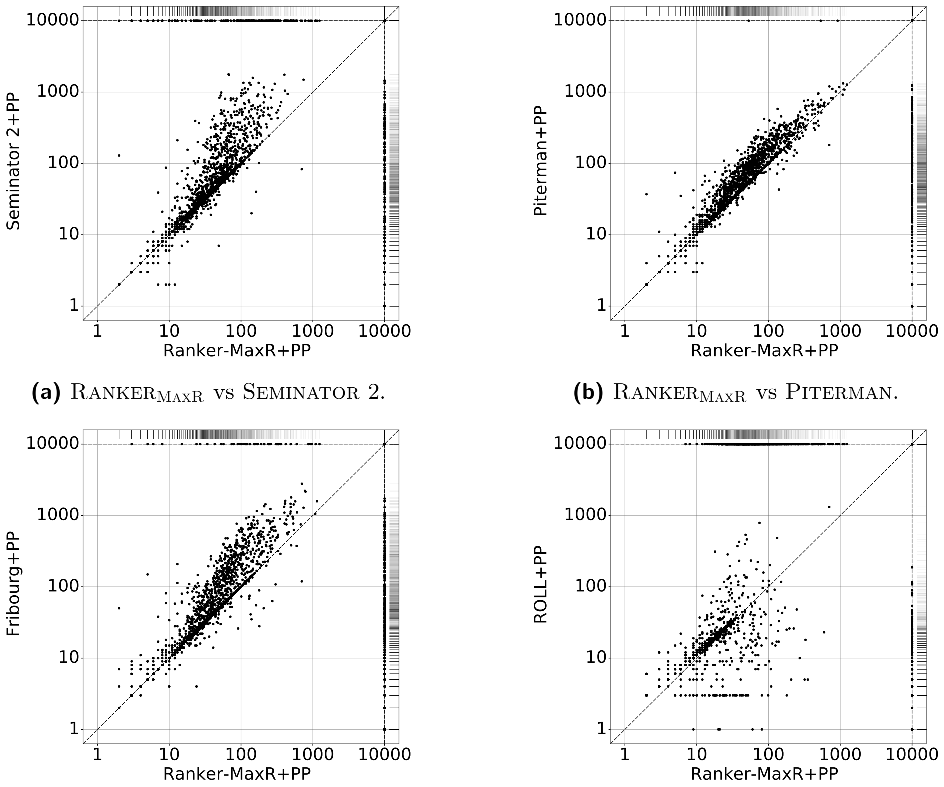
(The Purge$_{di}$ optimization is from [8].) Note that the two settings include all optimizations compatible with MaxRank and RankRestr respectively. Due to space constraints, we cannot give a detailed analysis of the effect of individual optimizations on the size of the obtained complement automaton. Let us, at least, give a bird's-eye view. The biggest effect has MaxRank, followed by Delay– their use is key to obtaining a small state space. The rest of the optimizations are less effective, but they still remove a significant number of states.

5.1 Comparison of Rank-Based Procedures

First, we evaluated how our optimizations reduce the generated state space, i.e., we compared the sizes of complemented BAs with no postprocessing. Such a use case represents applications like testing inclusion or equivalence of BAs, where postprocessing the output is irrelevant.

More precisely, we first compared the sizes of automata produced by our settings Ranker$_{\text{MaxR}}$ and Ranker$_{\text{RRestr}}$ to see which of them behaves better (cf. Figure 3a) and then we compared Ranker$_{\text{MaxR}}$, which had better results, with the Schewe$_{\text{RedAvgOut}}$ procedure implemented in GOAL (parameters `-m rank -tr -ro`). Scatter plots of the results are given in Figure 3b and summarizing statistics in the upper part of Table 1.

We note that although Ranker$_{\text{MaxR}}$ produces in the vast majority of cases (1,810) smaller automata than Ranker$_{\text{RRestr}}$, in a few cases (109) Ranker$_{\text{RRestr}}$ still outputs a smaller result (in 1 case this is due to the timeout of Ranker$_{\text{MaxR}}$). The comparison with Schewe$_{\text{RedAvgOut}}$ shows that our optimizations indeed have a profound effect on the size of the generated state space. Although the mean and maximum size of complements produced by



(a) Ranker$_{\text{MaxR}}$ vs Seminator 2.

(b) Ranker$_{\text{MaxR}}$ vs Piterman.

(c) Ranker$_{\text{MaxR}}$ vs Fribourg.

(d) Ranker$_{\text{MaxR}}$ vs ROLL.

Figure 4 Comparison of the sizes of the BAs constructed using our optimized rank-based construction and other approaches. Timeouts are on the dashed lines.

Ranker$_{\text{MaxR}}$ and Ranker$_{\text{RRestr}}$ are larger than those of Schewe$_{\text{RedAvgOut}}$, this is because for cases where the complement would be large, the run of Schewe$_{\text{RedAvgOut}}$ in GOAL timeouted before it could produce a result. Therefore, the median is a more meaningful indicator, and it is significantly (3–4×) lower for both Ranker$_{\text{MaxR}}$ and Ranker$_{\text{RRestr}}$.

5.2 Comparison with Other Approaches

Further, we evaluated the complements produced by Ranker$_{\text{MaxR}}$ and other approaches. In this setting, we focused on the size of the output BA *after* postprocessing (we, again, used `autfilt` with simplification `-high`; we denote this using "+PP"). We evaluated the following algorithms: Safra [32], its optimization Piterman [30] the optimization implemented in LTL2dstar [21], Fribourg [1], Spot (Redziejowski's algorithm [31]), ROLL's learning-based algorithm [25], and a semideterminization-based algorithm [3] in Seminator 2.

In Figure 4, we give scatter plots of selected comparisons; we omitted the results for Safra, Spot, and LTL2dstar, which on average performed slightly worse than Piterman. We give summarizing statistics in the lower part of Table 1 and the run times in Table 2.

Let us now discuss the data in the lower part of Table 1. In the left-hand side, we can see that the mean and median size of BAs obtained by Ranker$_{\text{MaxR}}$ are both the lowest with the exception of ROLL. ROLL implements a learning-based approach, which means that it works on the level of the *language* of the input BA instead of the *structure*. Therefore, it can often find a much smaller automaton than other approaches. Its practical time complexity,

Table 2 Run times of the tools [s].

method	mean	med.	std. dev
Ranker$_{\text{MaxR}}$	10.21	0.84	28.43
Ranker$_{\text{MaxR}}$+BO	9.40	3.03	16.00
Piterman ⚽	7.47	6.03	8.46
Safra ⚽	15.49	7.03	35.59
Spot	1.07	0.02	8.94
Fribourg ⚽	19.43	10.01	32.76
LTL2dstar	4.17	0.06	22.19
Seminator 2	11.41	0.37	34.97
ROLL	42.63	14.92	67.31

Table 3 Wins and losses for Ranker$_{\text{MaxR}}$+BO.

method	wins	(TO)	losses	(TO)
Piterman ⚽	1 160	(4)	112	(9)
Safra ⚽	1 255	(147)	222	(6)
Spot	985	(8)	328	(12)
Fribourg ⚽	1 076	(71)	287	(10)
LTL2dstar	1 208	(118)	272	(7)
Seminator 2	1 236	(333)	253	(5)
ROLL	1 923	(1 096)	360	(7)

however, seems to grow much faster with the number of states of the output BA than other approaches (cf. Table 2). Ranker$_{\text{MaxR}}$ by itself had more timeouts than other approaches, but when used with the BackOff strategy, is on par with Piterman and Spot.

In the right-hand side of Table 1, we give the numbers of times where Ranker$_{\text{MaxR}}$ gave strictly smaller and strictly larger outputs respectively. Here, we can see that the output of Ranker$_{\text{MaxR}}$ is often at least as small as the output of the other method (this is not in the table, but can be computed as $2,393 - \textbf{losses}$; the losses were caused mostly by timeouts; results with the BackOff strategy would increase the number even more) and often a strictly smaller one (the **wins** column). When comparing Ranker$_{\text{MaxR}}$ with *the best result of any other tool*, it obtained a *strictly smaller* BA in 539 cases (22.5 %) and a BA *at least as small* as the best result of any other tool in 1,518 cases (63.4 %). Lastly, we note that there were four BAs in the benchmark that *no tool* could complement and one BA that *only* Ranker$_{\text{MaxR}}$ was able to complement; there was no such a case for any other tool.

Let us now focus on the run times of the tools in Table 2. GOAL and ROLL are implemented in Java, which adds a significant overhead to the run time (e.g., the fastest run time of GOAL was 3.15 s; it is hard to predict how their performance would change if they were reimplemented in a faster language); the other approaches are implemented in C++.

BackOff. Our BackOff setting in the experiments used the set of constraints $\{(\mathit{StateSize}_1 = 9, \mathit{RankMax}_1 = 5), (\mathit{StateSize}_2 = 8, \mathit{RankMax}_2 = 6)\}$ and Piterman as the surrogate algorithm. The BackOff strategy was executed 873 times and managed to decrease the number of timeouts of Ranker$_{\text{MaxR}}$ from 360 to 17 (row Ranker$_{\text{MaxR}}$+BO in Table 1).

Discussion. The results of our experiments show that our optimizations are key to making rank-based complementation competitive to other approaches in practice. Furthermore, with the optimizations, the obtained procedure in the majority of cases produces a BA at least as small as a BA produced by any other approach, and in a large number of cases *the smallest* BA produced by any existing approach. We emphasize the usefulness of the BackOff heuristic: as there is no clear "best" complementation algorithm – different techniques having different strengths and weaknesses – knowing which technique to use for an input automaton is important in practice. In Table 3, we give a modification of the right-hand side of Table 1 giving wins and losses for Ranker$_{\text{MaxR}}$+BO. It seems that the combination of these two completely different algorithms yields a quite strong competitor.

6 Related Work

The problem of BA complementation has attracted researchers since Büchi's seminal work [7]. Since then, there have appeared several directions of BA complementation approaches. *Ramsey-based complementation* using Büchi's original argument, decomposing the language accepted by an automaton into a finite number of equivalence classes, was later improved in [6]. *Determinization-based complementation* was introduced by Safra in [32], later improved by Piterman in [30]. Determinization-based approaches convert an input BA into an equivalent intermediate deterministic automaton with different accepting condition (e.g. Rabin automaton) that can be easily complemented. The result is then converted back into a BA (often for the price of some blow-up). *Slice-based complementation* uses a reduced abstraction on a run tree to track the acceptance condition [39, 19]. *A learning-based approach* was presented in [25, 24]. A novel optimal complementation algorithm by Allred and Utes-Nitsche was presented in [1]. There are also specific approaches for complementation of special types of BAs, e.g., deterministic [23], semi-deterministic [3], or unambiguous [26]. *Semi-determinization based complementation* then uses a conversion of a standard BA to a semi-deterministic version [10] followed by its complementation [4].

Rank-based complementation, studied in [22, 15, 13, 33, 20], extends the subset construction for determinizing finite automata with additional information kept in each macrostate to track the acceptance condition of all runs of the input automaton. We have described the refinement of the basic procedure from [22] towards [13] and [33] in Section 3. The work in [15] contains optimizations of an alternative (sub-optimal) rank-based construction from [22] that goes through *alternating Büchi automata*. Furthermore, the work in [20] proposes an optimization of Schewe that in some cases produces smaller automata (the construction is not compatible with our optimizations). Rank-based construction can be optimized using simulation relations as shown in [8]. Here the direct and delayed simulation relations can be used to prune macrostates that are redundant for accepting a word or to saturate macrostates with simulation-smaller states.

References

1 Joël D. Allred and Ulrich Ultes-Nitsche. A simple and optimal complementation algorithm for Büchi automata. In *Proceedings of the Thirty third Annual IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 46–55. IEEE Computer Society Press, July 2018.

2 Tomáš Babiak, František Blahoudek, Alexandre Duret-Lutz, Joachim Klein, Jan Křetínský, David Müller, David Parker, and Jan Strejček. The Hanoi omega-automata format. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 479–486. Springer, 2015. doi:10.1007/978-3-319-21690-4_31.

3 Frantisek Blahoudek, Matthias Heizmann, Sven Schewe, Jan Strejcek, and Ming-Hsien Tsai. Complementing semi-deterministic Büchi automata. In Marsha Chechik and Jean-François Raskin, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 22nd International Conference, TACAS 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9636 of *Lecture Notes in Computer Science*, pages 770–787. Springer, 2016. doi:10.1007/978-3-662-49674-9_49.

4 František Blahoudek, Alexandre Duret-Lutz, and Jan Strejček. Seminator 2 can complement generalized Büchi automata via improved semi-determinization. In *Proceedings of the 32nd International Conference on Computer-Aided Verification (CAV'20)*, volume 12225 of *Lecture Notes in Computer Science*, pages 15–27. Springer, 2020. doi:10.1007/978-3-030-53291-8_2.

5 Bernard Boigelot, Sébastien Jodogne, and Pierre Wolper. On the use of weak automata for deciding linear arithmetic with integer and real variables. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, volume 2083 of *Lecture Notes in Computer Science*, pages 611–625. Springer, 2001. `doi:10.1007/3-540-45744-5_50`.

6 Stefan Breuers, Christof Löding, and Jörg Olschewski. Improved Ramsey-based Büchi complementation. In *Proc. of FOSSACS'12*, pages 150–164. Springer, 2012.

7 J. Richard Büchi. On a decision method in restricted second order arithmetic. In *Proc. of International Congress on Logic, Method, and Philosophy of Science 1960*. Stanford Univ. Press, Stanford, 1962.

8 Yu-Fang Chen, Vojtech Havlena, and Ondrej Lengál. Simulations in rank-based Büchi automata complementation. In Anthony Widjaja Lin, editor, *Programming Languages and Systems – 17th Asian Symposium, APLAS 2019, Nusa Dua, Bali, Indonesia, December 1-4, 2019, Proceedings*, volume 11893 of *Lecture Notes in Computer Science*, pages 447–467. Springer, 2019. `doi:10.1007/978-3-030-34175-6_23`.

9 Yu-Fang Chen, Matthias Heizmann, Ondrej Lengál, Yong Li, Ming-Hsien Tsai, Andrea Turrini, and Lijun Zhang. Advanced automata-based algorithms for program termination checking. In Jeffrey S. Foster and Dan Grossman, editors, *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2018, Philadelphia, PA, USA, June 18-22, 2018*, pages 135–150. ACM, 2018. `doi:10.1145/3192366.3192405`.

10 Costas Courcoubetis and Mihalis Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 338–345. IEEE Computer Society, 1988. `doi:10.1109/SFCS.1988.21950`.

11 Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Étienne Renault, and Laurent Xu. Spot 2.0 – a framework for LTL and ω-automata manipulation. In Cyrille Artho, Axel Legay, and Doron Peled, editors, *Automated Technology for Verification and Analysis*, pages 122–129, Cham, 2016. Springer International Publishing.

12 Seth Fogarty and Moshe Y. Vardi. Büchi complementation and size-change termination. In *Proc. of TACAS'09*, pages 16–30. Springer, 2009.

13 Ehud Friedgut, Orna Kupferman, and Moshe Vardi. Büchi complementation made tighter. *International Journal of Foundations of Computer Science*, 17:851–868, 2006.

14 Patrice Godefroid. Using partial orders to improve automatic verification methods. In Edmund M. Clarke and Robert P. Kurshan, editors, *Computer Aided Verification, 2nd International Workshop, CAV '90, New Brunswick, NJ, USA, June 18-21, 1990, Proceedings*, volume 531 of *Lecture Notes in Computer Science*, pages 176–185. Springer, 1990. `doi:10.1007/BFb0023731`.

15 Sankar Gurumurthy, Orna Kupferman, Fabio Somenzi, and Moshe Y. Vardi. On complementing nondeterministic Büchi automata. In Daniel Geist and Enrico Tronci, editors, *Correct Hardware Design and Verification Methods, 12th IFIP WG 10.5 Advanced Research Working Conference, CHARME 2003, L'Aquila, Italy, October 21-24, 2003, Proceedings*, volume 2860 of *Lecture Notes in Computer Science*, pages 96–110. Springer, 2003. `doi:10.1007/978-3-540-39724-3_10`.

16 Vojtěch Havlena and Ondřej Lengál. Reducing (to) the ranks: Efficient rank-based Büchi automata complementation (technical report). *CoRR*, abs/2010.07834, 2020. `arXiv:2010.07834`.

17 Vojtěch Havlena and Ondřej Lengál. RANKER, 2021. URL: `https://github.com/vhavlena/ba-inclusion`.

18 Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Termination analysis by learning terminating programs. In *Proc. of CAV'14*, pages 797–813. Springer, 2014.

19 Detlef Kähler and Thomas Wilke. Complementation, disambiguation, and determinization of Büchi automata unified. In *Proc. of ICALP'08*, pages 724–735. Springer, 2008.

20 Hrishikesh Karmarkar and Supratik Chakraborty. On minimal odd rankings for Büchi complementation. In Zhiming Liu and Anders P. Ravn, editors, *Automated Technology for Verification and Analysis, 7th International Symposium, ATVA 2009, Macao, China, October 14-16, 2009. Proceedings*, volume 5799 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2009. doi:10.1007/978-3-642-04761-9_18.

21 Joachim Klein and Christel Baier. On-the-fly stuttering in the construction of deterministic *omega* -automata. In Jan Holub and Jan Zdárek, editors, *Implementation and Application of Automata, 12th International Conference, CIAA 2007, Prague, Czech Republic, July 16-18, 2007, Revised Selected Papers*, volume 4783 of *Lecture Notes in Computer Science*, pages 51–61. Springer, 2007. doi:10.1007/978-3-540-76336-9_7.

22 Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001. doi:10.1145/377978.377993.

23 Robert P. Kurshan. Complementing deterministic Büchi automata in polynomial time. *J. Comput. Syst. Sci.*, 35(1):59–71, 1987. doi:10.1016/0022-0000(87)90036-5.

24 Yong Li, Xuechao Sun, Andrea Turrini, Yu-Fang Chen, and Junnan Xu. ROLL 1.0: ω-regular language learning library. In Tomás Vojnar and Lijun Zhang, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 25th International Conference, TACAS 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings, Part I*, volume 11427 of *Lecture Notes in Computer Science*, pages 365–371. Springer, 2019. doi:10.1007/978-3-030-17462-0_23.

25 Yong Li, Andrea Turrini, Lijun Zhang, and Sven Schewe. Learning to complement Büchi automata. In *Proc. of VMCAI'18*, pages 313–335. Springer, 2018.

26 Yong Li, Moshe Y. Vardi, and Lijun Zhang. On the power of unambiguity in Büchi complementation. In Jean-Francois Raskin and Davide Bresolin, editors, Proceedings 11th International Symposium on *Games, Automata, Logics, and Formal Verification,* Brussels, Belgium, September 21-22, 2020, volume 326 of *Electronic Proceedings in Theoretical Computer Science*, pages 182–198. Open Publishing Association, 2020. doi:10.4204/EPTCS.326.12.

27 Richard Mayr and Lorenzo Clemente. Advanced automata minimization. In *Proc. of POPL'13*, pages 63–74, 2013.

28 Max Michel. Complementation is more difficult with automata on infinite words. *CNET, Paris*, 15, 1988.

29 Doron A. Peled. All from one, one for all: on model checking using representatives. In Costas Courcoubetis, editor, *Computer Aided Verification, 5th International Conference, CAV '93, Elounda, Greece, June 28 - July 1, 1993, Proceedings*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer, 1993. doi:10.1007/3-540-56922-7_34.

30 Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. of LICS'06*, pages 255–264. IEEE, 2006.

31 Roman R. Redziejowski. An improved construction of deterministic omega-automaton using derivatives. *Fundam. Informaticae*, 119(3-4):393–406, 2012. doi:10.3233/FI-2012-744.

32 Shmuel Safra. On the complexity of ω-automata. In *Proc. of FOCS'88*, pages 319–327. IEEE, 1988.

33 Sven Schewe. Büchi complementation made tight. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPIcs*, pages 661–672. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009. doi:10.4230/LIPIcs.STACS.2009.1854.

34 Prasad A. Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for Büchi automata with applications to temporal logic. In *Automata, Languages and Programming*, pages 465–474. Springer, 1985.

35 Deian Tabakov and Moshe Y. Vardi. Experimental evaluation of classical automata constructions. In *Proc. of LPAR'05*, pages 396–411. Springer, 2005.

36 Ming-Hsien Tsai, Seth Fogarty, Moshe Y. Vardi, and Yih-Kuen Tsay. State of Büchi complementation. In Michael Domaratzki and Kai Salomaa, editors, *Implementation and Application of Automata*, pages 261–271, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

37 Ming-Hsien Tsai, Yih-Kuen Tsay, and Yu-Shiang Hwang. GOAL for games, omega-automata, and logics. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, pages 883–889, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

38 Antti Valmari. Stubborn sets for reduced state space generation. In Grzegorz Rozenberg, editor, *Advances in Petri Nets 1990*, pages 491–515, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

39 Moshe Y. Vardi and Thomas Wilke. Automata: From logics to algorithms. *Logic and Automata*, 2:629–736, 2008.

40 Qiqi Yan. Lower bounds for complementation of ω-automata via the full automata technique. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, pages 589–600, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

41 Qiqi Yan. Lower bounds for complementation of ω-automata via the full automata technique. In *Proc. of ICALP'06*, pages 589–600. Springer, 2006.

Inclusion Testing of Büchi Automata Based on Well-Quasiorders

Kyveli Doveri
IMDEA Software Institute, Madrid, Spain
Universidad Politécnica de Madrid, Spain

Pierre Ganty
IMDEA Software Institute, Madrid, Spain

Francesco Parolini
Sorbonne Université, Paris, France

Francesco Ranzato
University of Padova, Italy

Abstract

We introduce an algorithmic framework to decide whether inclusion holds between languages of infinite words over a finite alphabet. Our approach falls within the class of Ramsey-based methods and relies on a least fixpoint characterization of ω-languages leveraging ultimately periodic infinite words of type uv^ω, with u a finite prefix and v a finite period of an infinite word. We put forward an inclusion checking algorithm between Büchi automata, called BAInc, designed as a complete abstract interpretation using a pair of well-quasiorders on finite words. BAInc is quite simple: it consists of two least fixpoint computations (one for prefixes and the other for periods) manipulating finite sets (of pairs) of states compared by set inclusion, so that language inclusion holds when the sets (of pairs) of states of the fixpoints satisfy some basic conditions. We implemented BAInc in a tool called BAIT that we experimentally evaluated against the state-of-the-art. We gathered, in addition to existing benchmarks, a large number of new case studies stemming from program verification and word combinatorics, thereby significantly expanding both the scope and size of the available benchmark set. Our experimental results show that BAIT advances the state-of-the-art on an overwhelming majority of these benchmarks. Finally, we demonstrate the generality of our algorithmic framework by instantiating it to the inclusion problem of Büchi pushdown automata into Büchi automata.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Büchi (Pushdown) Automata, ω-Language Inclusion, Well-quasiorders

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.3

Supplementary Material *Software (Source Code)*: `https://github.com/parof/bait`
archived at `swh:1:dir:0085d9456cf1300c44386339a996c9453dca17c2`

Funding *Pierre Ganty*: Partially supported by the Madrid regional project S2018/TCS-4339 BLOQUES, the Spanish project PGC2018-102210-B-I00 BOSCO and the Ramón y Cajal fellowship RYC-2016-20281.
Francesco Ranzato: Partially funded by *University of Padova*, under the SID2018 project "Analysis of STatic Analyses (ASTA)"; *Italian Ministry of University and Research*, under the PRIN2017 project no. 201784YSZ5 "AnalysiS of PRogram Analyses (ASPRA)"; *Facebook Research*, under a "Probability and Programming Research Award".

Acknowledgements We thank: Richard Mayr, Lorenzo Clemente, Parosh Abdulla and Lukáš Holík for their insights and help with RABIT; Ming-Hsien Tsai for his help with GOAL; Reed Oei for the Pecan benchmarks; Matthias Heizmann and Daniel Dietsch for the Ultimate Automizer benchmarks.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 3; pp. 3:1–3:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Deciding whether a formal language contains another one is a fundamental problem with diverse applications ranging from automata-based verification to compiler construction [6, 13, 25, 42]. In this work, we deal with the inclusion problem for ω-languages, namely languages of words of infinite length (ω-words) over a finite alphabet. In particular, we are interested in the case of ω-regular languages, which is known to be PSPACE-complete [26], and in the inclusion of ω-context-free languages into ω-regular, which is known to be EXPTIME-complete [21, 32].

1.1 Main Contributions

We put forward a number of language inclusion algorithms that are systematically designed from an abstraction-based perspective of the inclusion problem. Our starting point was a recent abstract interpretation-based algorithmic framework for the inclusion problem for languages of finite words [15, 16]. Extending this framework to ω-words raises several challenges. First, the finite word case crucially relies on least fixpoint characterizations of languages which we are not aware of for languages of ω-words (while greatest fixpoint characterizations exist). The second challenge is to define suitable abstractions for languages of ω-words and effective representations thereof.

We overcome the first challenge by reducing the inclusion problem for ω-languages to an equivalent inclusion problem between their so-called *ultimately periodic* subsets. The ultimately periodic subset of an ω-language L consists of those ω-words of the form $uv^\omega \in L$, where u and v are finite words referred to as, resp., a *prefix* and a *period* of an ω-word. It turns out that an underlying Büchi (pushdown) automaton accepting L enables a least fixpoint characterization of the ultimately periodic subset of L. To guarantee convergence in finitely many Kleene iterations of such a least fixpoint, we resort to a conceptually simple approach based on abstract interpretation [8]. Roughly speaking, we define over-approximating abstractions of sets of finite words which “enlarge” these sets with new words picked according to a quasiorder relation on finite words. Our abstractions rely on two distinct quasiorder relations which, resp., enlarge the sets of prefixes and periods of an ultimately periodic set representing an ω-language. The quasiorders inducing our abstractions have to satisfy two basic properties: (1) to be well-quasiorders to guarantee finite convergence of fixpoint computations; (2) some monotonicity conditions w.r.t. word concatenation in order to yield a sound and complete inclusion algorithm (soundness holds for mere quasiorders). Once the abstract least fixpoint has been computed, an inclusion check $L \subseteq^? M$ reduces to a finite number of tests $uv^\omega \in^? M$ for finitely many prefixes u and periods v taken from the abstract least fixpoint representing L. We introduce different well-quasiorders to be used in our inclusion algorithm and we show that using distinct well-quasiorder-based abstractions for prefixes and periods pays off.

For a language inclusion check $L \subseteq M$, where L and M are accepted by Büchi automata, some quasiorders enable a further abstraction step where finite words are abstracted by states relating these words in the underlying Büchi automaton accepting M, and this correspondingly defines a purely “state-based” inclusion algorithm that operates on automaton states only. We further demonstrate the generality of our algorithmic framework by instantiating it to the inclusion problem of Büchi pushdown automata into Büchi automata.

We implemented our language inclusion algorithm in a tool called BAIT (Büchi Automata Inclusion Tester) [10]. We put together an extensive suite of benchmarks [11], notably verification tasks as defined by the RABIT tool [1, 2], logical implication tasks in word combinatorics as defined by the Pecan theorem prover [34], and termination tasks as defined by Ultimate Automizer [18]. We conducted an experimental comparison of BAIT against some state-of-the-art language inclusion checking tools: GOAL [41], HKCω [24], RABIT [37, 7] and ROLL [27]. The experimental results show that BAIT advances the state-of-the-art of the tools for checking inclusion of ω-languages on an overwhelming majority of benchmarks.

1.2 Related Works

Due to space constraints, we limit our discussion to Ramsey-based algorithms, as our inclusion procedure, and to methods based on automata complementation. Kuperberg et al. [24] also reduce the language equivalence problem over Büchi automata to that of their ultimately periodic subsets. A further commonality is that the algorithm of [24] handles prefixes and periods differently: for the prefixes they leverage a state-of-the-art up-to congruence algorithm [3], while up-to congruences are not used for the periods[1]. Fogarty and Vardi [14] for the universality problem, and later Abdulla et al. [1, 2] for the inclusion problem between languages accepted by Büchi automata, all reduce their decision problems to the ultimately periodic subsets. Their approach is based on a partition of nonempty words whose blocks are represented and manipulated through so-called supergraphs. The equivalence relation underlying their partition can be obtained from one of our quasiorders. Moreover, by equipping their supergraphs with a subsumption order [2, Def. 6], they define a relation which coincides with one of our quasiorders. Hofmann and Chen [20], whose approach based on abstract interpretation inspired our work, also tackle the inclusion problem for ω-languages. They construct an abstract (finite) lattice using the same equivalence relation which is derived from a given Büchi atomaton, and define a Galois connection between it and the (infinite) lattice of languages of infinite words. However, they do not relax this relation into a quasiorder. Finally, the complementation-based approaches reduce language inclusion to a language emptiness check by using intersection and an explicit complementation of a Büchi automaton. Despite that there are Büchi automata of size n whose complement cannot be represented with less than $n!$ states [33], algorithms to complement Büchi automata have been defined, implemented and are effective in practice [40]. In our approach, explicit complementation is avoided altogether.

2 Overview

We assume familiarity with the basics of language theory (see, e.g., [22, 35]). Throughout the paper, we fix Σ to be a finite nonempty alphabet. Furthermore, let ϵ denote the empty word, Σ^* the set of finite words over Σ, $\Sigma^+ \triangleq \Sigma^* \setminus \{\epsilon\}$, Σ^ω the set of infinite words (or ω-words) over Σ, $|w| \in \mathbb{N}$ denote the length of $w \in \Sigma^*$. The *ultimately periodic words* are the words $\xi \in \Sigma^\omega$ such that $\xi = uv^\omega$ for some finite *prefix* $u \in \Sigma^*$ and some finite *period* $v \in \Sigma^+$. Given $L \subseteq \Sigma^\omega$, we associate pairs of finite words to ultimately periodic words and define

$$I_L \triangleq \{(u, v) \in \Sigma^* \times \Sigma^+ \mid uv^\omega \in L\} \ .$$

In the following we give an outline of our approach. Given two ω-languages L and M such that the inclusion check reduces to that of their ultimately periodic words, i.e. $L \subseteq M \Leftrightarrow I_L \subseteq I_M$ holds, we reduce the inclusion problem $L \subseteq M$ to finitely many membership queries in the candidate "larger" language M.

[1] In the technical report thereof, the authors work out up-to union and up-to equivalence reasoning for periods but not their combination (up-to congruence).

A quasiorder (qo) relation on a set S is a reflexive and transitive binary relation on S. Any qo $\leq \subseteq S \times S$ induces a map $\rho_{\leq} : \wp(S) \to \wp(S)$ defined by $\rho_{\leq}(X) \triangleq \{y \in S \mid \exists x \in X,\, x \leq y\}$, which turns out to be a closure operator on the complete lattice $\langle \wp(S), \subseteq \rangle$. Let us recall that a closure operator is a monotone ($X \subseteq X' \Rightarrow \rho(X) \subseteq \rho(X')$), idempotent ($\rho(X) = \rho(\rho(X))$), and increasing ($X \subseteq \rho(X)$) map. Given $X \in \wp(S)$, the set $\rho_{\leq}(X)$ is called the *upward closure* of X w.r.t. $\leq$. We say that a qo relation $\preceq$ on $\Sigma^* \times \Sigma^+$ *preserves* I_M if $\rho_{\preceq}(I_M) = I_M$ holds. Given a qo $\preceq$ that preserves I_M, since $\rho_{\preceq}$ is monotone and increasing, we have that:

$$L \subseteq M \iff I_L \subseteq I_M \iff \rho_{\preceq}(I_L) \subseteq I_M \ . \qquad (1)$$

A qo $\leq$ is a well-quasiorder (wqo) if for any upward closure $\rho_{\leq}(X)$ there is a finite subset $X' \subseteq_{\text{fin}} X$ such that $\rho_{\leq}(X) = \rho_{\leq}(X')$. Hence, if a relation $\preceq$ on $\Sigma^* \times \Sigma^+$ is a wqo then there exists a finite subset $T \subseteq_{\text{fin}} I_L$ such that $\rho_{\preceq}(T) = \rho_{\preceq}(I_L)$. By exploiting the properties of closures, this reduces the inclusion check to finitely many membership queries in M:

$$L \subseteq M \iff \rho_{\preceq}(T) \subseteq I_M \iff T \subseteq I_M \iff \forall (u,v) \in T,\, uv^\omega \in M \ . \qquad (2)$$

Following this approach, we design inclusion algorithms in the cases where both languages L and M are ω-regular and where the “left” language L is ω-context-free and the “right” language M is ω-regular. In Section 3, we define wqos that preserve I_M as required by (1). Section 4 gives a detailed account of each step so as to end up designing our inclusion algorithms. Section 5 shows how to obtain algorithms deciding $L(\mathcal{A}) \subseteq L(\mathcal{B})$ and $L(\mathcal{P}) \subseteq L(\mathcal{B})$, where $\mathcal{A}$ and $\mathcal{B}$ are Büchi automata and $\mathcal{P}$ is a Büchi pushdown automaton, by reasoning exclusively on the automata states/configurations. Section 6 describes the experimental results of our implementation BAIT.

3 Well-Quasiorders for ω-Regular Languages

The equivalence (1) holds because the qo $\preceq$ on $\Sigma^* \times \Sigma^+$ is such that $\rho_{\preceq}(I_M) = I_M$. In the following, we focus on pairs of qos $\leq$ and $\preccurlyeq$ on, resp., Σ^* and Σ^+, such that their product relation $\leq \times \preccurlyeq$ on $\Sigma^* \times \Sigma^+$ preserves I_M, i.e., $\rho_{\leq\times\preccurlyeq}(I_M) = I_M$ holds. We define different pairs of qos preserving I_M and show how they compare. All these qos are *well-quasiorders* and *right-monotonic*. The first property guarantees the existence of a finite representation for I_L and the convergence after finitely many steps of the fixpoint computations, while the second property ultimately yields a (sound and) complete inclusion algorithm.

A qo $\leq$ on Σ^* is *left-monotonic* (*right-monotonic*) if

$$\forall u,v,w \in \Sigma^*,\ u \leq v \ \Rightarrow\ wu \leq wv \ \ (uw \leq vw) \ ,$$

while $\leq$ is monotonic if it is both left- and right-monotonic. Given any relation $R \subseteq X \times X$, $R^* \triangleq \bigcup_{n\in\mathbb{N}} R^n$ denotes its reflexive and transitive closure.

A Büchi automaton (BA) on an alphabet Σ is a tuple $\mathcal{A} = (Q, \delta, i, F)$ where Q is a finite set of states including a unique initial state $i \in Q$, $\delta \colon Q \times \Sigma \to \wp(Q)$ is a transition function, and $F \subseteq Q$ is a subset of final states. We write a transition $q \xrightarrow{a} q'$ when $q' \in \delta(q,a)$ and lift this relation to finite words by transitive and reflexive closure, thus writing $q \xrightarrow{u}{}^{*} q'$ with $u \in \Sigma^*$. We write $q \xrightarrow{u}_{F}{}^{*} q'$ if there exists $q_f \in F$ and $u_1, u_2 \in \Sigma^*$ such that $q \xrightarrow{u_1}{}^{*} q_f$, $q_f \xrightarrow{u_2}{}^{*} q'$ and $u = u_1u_2$. The language of finite words accepted by $\mathcal{A}$ is $L^*(\mathcal{A}) \triangleq \{u \in \Sigma^* \mid i \xrightarrow{u}{}^{*} q,\, q \in F\}$. A trace of $\mathcal{A}$ on an ω-word $w = a_0a_1 \cdots \in \Sigma^\omega$ is an infinite sequence $q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \cdots$, which is called initial when $q_0 = i$ and fair when $q_j \in F$ for infinitely many j's. The ω-language accepted by $\mathcal{A}$ is $L^\omega(\mathcal{A}) \triangleq \{w \in \Sigma^\omega \mid$ there exists an initial and fair trace on $w\}$. An ω-language $L \subseteq \Sigma^\omega$ is *ω-regular* if $L = L^\omega(\mathcal{A})$ for some BA $\mathcal{A}$.

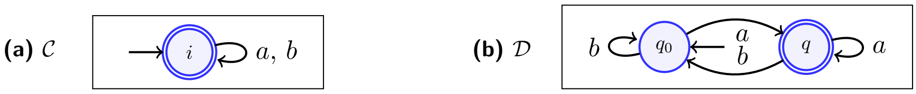


Figure 1 Büchi automata $\mathcal{C}$ and $\mathcal{D}$ over the alphabet $\Sigma = \{a, b\}$.

State-Based Quasiorders. We define quasiorders that compare words in Σ^* based on the states of a BA $\mathcal{A} = (Q, \delta, i, F)$. To do so, we associate with each word $u \in \Sigma^*$ its *context* $c^{\mathcal{A}}[u] \subseteq Q^2$ and *final context* $f^{\mathcal{A}}[u] \subseteq Q^2$ in $\mathcal{A}$ as follows:

$$c^{\mathcal{A}}[u] \triangleq \{(q, q') \in Q^2 \mid q \xrightarrow{u}{}^{*} q'\} \ ,$$
$$f^{\mathcal{A}}[u] \triangleq \{(q, q') \in Q^2 \mid q \xrightarrow{u}{}_{F}^{*} q'\} \ .$$

We also define the *successor set* $s^{\mathcal{A}}[u] \subseteq Q$ in $\mathcal{A}$ through a word $u \in \Sigma^*$ as follows:

$$s^{\mathcal{A}}[u] \triangleq \{q \in Q \mid i \xrightarrow{u}{}^{*} q\} \ .$$

Based on this, we define the following qos on words in Σ^*:

$$u \leq^{\mathcal{A}} v \stackrel{\triangle}{\Leftrightarrow} s^{\mathcal{A}}[u] \subseteq s^{\mathcal{A}}[v] \ ,$$
$$u \trianglelefteq^{\mathcal{A}} v \stackrel{\triangle}{\Leftrightarrow} c^{\mathcal{A}}[u] \subseteq c^{\mathcal{A}}[v] \ ,$$
$$u \preccurlyeq^{\mathcal{A}} v \stackrel{\triangle}{\Leftrightarrow} u \trianglelefteq^{\mathcal{A}} v \land f^{\mathcal{A}}[u] \subseteq f^{\mathcal{A}}[v] \ .$$

▶ **Example 3.1.** Consider the BA $\mathcal{D}$ in Fig. 1 (b). Since for all $u \in \Sigma^*$, $s^{\mathcal{D}}[ua] = \{q\}$ and $s^{\mathcal{D}}[ub] = s^{\mathcal{D}}[\epsilon] = \{q_0\}$, we have that $u \leq^{\mathcal{D}} v$ iff either $u, v \in \Sigma^* a$ or $u, v \in \Sigma^* b \cup \{\epsilon\}$. Similarly, we find that $u \trianglelefteq^{\mathcal{D}} v$ iff either $u, v \notin \{\epsilon\}$ and $u \leq^{\mathcal{D}} v$, or $u, v \in \{\epsilon\}$. For $u \in \Sigma^*$ we have $f^{\mathcal{D}}[ua] = \{(q_0, q), (q, q)\}$. For $u \in \Sigma^* \setminus b^*$ we have $f^{\mathcal{D}}[ub] = \{(q_0, q_0), (q, q_0)\}$ and $f^{\mathcal{D}}[b^k] = \{(q, q_0)\}$, for any $k \geq 1$. As for the empty word, $f^{\mathcal{D}}[\epsilon] = \{(q, q)\}$. Hence, for all $u, v \in \Sigma^*$, it turns out that $u \preccurlyeq^{\mathcal{C}} v$ holds iff one of the following four cases holds: (i) $u, v \in \Sigma^* a$; (ii) $u \in \Sigma^* b$ and $v \in \Sigma^* b \backslash b^*$; (iii) $u, v \in b^+$; (iv) $u, v \in \{\epsilon\}$. ⌟

The qos $\trianglelefteq^{\mathcal{A}}$ and $\leq^{\mathcal{A}}$ appeared in [15] while $\preccurlyeq^{\mathcal{A}}$ was obtained by relaxing an equivalence defined in [20]. By definition, we have that $\preccurlyeq^{\mathcal{A}} \subseteq \trianglelefteq^{\mathcal{A}}$, and since $q \in s^{\mathcal{A}}[u]$ iff $(i, q) \in c^{\mathcal{A}}[u]$, we deduce that $\preccurlyeq^{\mathcal{A}} \subseteq \trianglelefteq^{\mathcal{A}} \subseteq \leq^{\mathcal{A}}$ holds. Since Q is a finite set, it turns out that all these three state-based qos are indeed wqos. It is easily seen that $\trianglelefteq^{\mathcal{A}}$ and $\preccurlyeq^{\mathcal{A}}$ are monotonic and that $\leq^{\mathcal{A}}$ is right-monotonic. Finally, we turn to the preservation property with respect to $I_{L^\omega(\mathcal{A})}$. Let $(u, v) \in I_{L^\omega(\mathcal{A})}$. Then, $uv^\omega \in L^\omega(\mathcal{A})$ and there is an initial fair trace of $\mathcal{A}$ on uv^ω. Hence, we can find two states $p, q \in Q$ and two integers $n, m \geq 1$ such that $i \xrightarrow{u}{}^{*} p$, $p \xrightarrow{v^n}{}^{*} q$ and $q \xrightarrow{v^m}{}_{F}^{*} q$. Let $(s, t) \in \Sigma^* \times \Sigma^+$ be such that $u \leq^{\mathcal{A}} s$ and $v \preccurlyeq^{\mathcal{A}} t$. By monotonicity of $\preccurlyeq^{\mathcal{A}}$, we deduce that $v^k \preccurlyeq^{\mathcal{A}} t^k$ holds for all $k \in \mathbb{N}$. Hence, by definition of the state-based qos, we also have $i \xrightarrow{s}{}^{*} p$, $p \xrightarrow{t^n}{}^{*} q$ and $q \xrightarrow{t^m}{}_{F}^{*} q$. Therefore, $(s, t) \in I_{L^\omega(\mathcal{A})}$ holds. The argument remains the same if $u \trianglelefteq^{\mathcal{A}} s$ or $u \preccurlyeq^{\mathcal{A}} s$, so that we conclude that the pair $\leq^{\mathcal{A}}, \preccurlyeq^{\mathcal{A}}$, as well as the pairs $\trianglelefteq^{\mathcal{A}}, \preccurlyeq^{\mathcal{A}}$ and $\preccurlyeq^{\mathcal{A}}, \preccurlyeq^{\mathcal{A}}$ are pairs of wqos preserving $I_{L^\omega(\mathcal{A})}$.

4 An Algorithmic Framework for Checking Inclusion

We start with the ω-regular $\subseteq$ ω-regular case and then leverage the generality of our algorithmic framework to tackle the ω-context-free $\subseteq$ ω-regular case in Section 4.2.

4.1 Language Inclusion ω-regular $\subseteq$ ω-regular

Let us first recall the following fundamental theorem for languages of ω-words.

▶ **Theorem 4.1** ([5]). *The equivalence $L \subseteq M \iff I_L \subseteq I_M$ holds for all ω-regular languages $L, M \subseteq \Sigma^\omega$.*

Fix an ω-regular language M, a pair $\leq, \preccurlyeq$ of right-monotonic wqos on, resp., Σ^* and Σ^+, with $\rho_{\leq\times\preccurlyeq}(I_M) = I_M$, as given in Section 3, and a BA $\mathcal{A} = (Q, \delta, i_\mathcal{A}, F)$ such that $L = L^\omega(\mathcal{A})$.

A Representation for the Ultimately Periodic Words of L. We slightly generalize the approach presented in Section 2 and represent the ultimately periodic words of L by a subset $S \subseteq I_L$ such that $\{uv^\omega \mid (u,v) \in S\} = \{uv^\omega \mid (u,v) \in I_L\}$ holds, so that $L \subseteq M \Leftrightarrow S \subseteq I_M$ holds. The definition of such a subset S representing I_L relies on the following result.

▶ **Lemma 4.2.** *Let $\mathcal{A} = (Q, \delta, i_\mathcal{A}, F)$ be a BA. Then, $uv^\omega \in L^\omega(\mathcal{A})$ iff there exist $p \in F$, $u' \in \Sigma^*$, $v' \in \Sigma^+$ such that $uv^\omega = u'v'^\omega$, $i_\mathcal{A} \xrightarrow{u'}{}^* p$ and $p \xrightarrow{v'}{}^* p$.*

For each pair of states $q, q' \in Q$, we define the automaton $\mathcal{A}^q_{q'} \triangleq (Q, \delta, q, \{q'\})$. By Lemma 4.2, it turns out that the ultimately periodic words generated by the pairs of finite words in

$$S_\mathcal{A} \triangleq \textstyle\bigcup_{p\in F} L^*(\mathcal{A}^{i_\mathcal{A}}_p) \times (L^*(\mathcal{A}^p_p)\backslash\{\epsilon\})$$

coincide with the ultimately periodic words of $L^\omega(\mathcal{A})$. Hence, by reasoning as in Section 2, it turns out that:

$$L^\omega(\mathcal{A}) \subseteq M \iff S_\mathcal{A} \subseteq I_M \iff \rho_{\leq\times\preccurlyeq}(S_\mathcal{A}) \subseteq I_M \quad . \tag{1'}$$

Fixpoint Characterization of $S_\mathcal{A}$. For a function $f : X \to X$ over a set X and for all $n \in \mathbb{N}$, the n-th power $f^n : X \to X$ of f is inductively defined as usual: $f^0 \triangleq \lambda x.x$; $f^{n+1} \triangleq f \circ f^n$. The denumerable sequence of *Kleene iterates* of f starting from an initial value $a \in X$ is given by $\{f^n(a)\}_{n\in\mathbb{N}}$. This sequence finitely converges to some $f^k(a)$, with $k \in \mathbb{N}$, when for all $n \geq k$, $f^n(a) = f^k(a)$. Let us recall that when X is a directed-complete partial order with bottom $\bot$ and f is monotone, if the Kleene iterates starting from the bottom $\{f^n(\bot)\}_{n\in\mathbb{N}}$ finitely converge to some $f^k(\bot)$ then $f^k(\bot)$ is the least fixpoint of f, denoted by $\operatorname{lfp} f$.

Given $X \in \wp(\Sigma^*)^Q$, we define

$$\mathrm{Post}_\mathcal{A}(X) \triangleq \langle \textstyle\bigcup_{a\in\Sigma, q\in\delta(q',a)} X_{q'}a\rangle_{q\in Q} \in \wp(\Sigma^*)^Q \quad ,$$

where, for all $q \in Q$, X_q denotes the q-indexed component of the vector X. In turn, for each $p \in F$, we define the maps

$$P_\mathcal{A} \triangleq \lambda X.\langle\{\epsilon \mid q = i_\mathcal{A}\} \cup \ (\mathrm{Post}_\mathcal{A}(X))_q\rangle_{q\in Q} \quad ,$$
$$R_{\mathcal{A},p} \triangleq \lambda X.\langle\{a \in \Sigma \mid q \in \delta(p,a)\} \cup \ (\mathrm{Post}_\mathcal{A}(X))_q\rangle_{q\in Q} \quad ,$$

which allows us to give the following least fixpoint characterization of $S_\mathcal{A}$.

▶ **Lemma 4.3.** $S_\mathcal{A} = \bigcup_{p\in F}(\operatorname{lfp} P_\mathcal{A})_p \times (\operatorname{lfp} R_{\mathcal{A},p})_p$.

▶ **Example 4.4.** Consider the BA $\mathcal{C}$ in Fig. 1. Since $L^*(\mathcal{C}^i_i) = \{a,b\}^*$, we have that $S_\mathcal{C} = \{a,b\}^* \times \{a,b\}^+$. Since $\mathcal{C}$ has only one state, vectors have dimension one. We have that $P_\mathcal{C} = \lambda X.\{\epsilon\} \cup Xa \cup Xb$ and $R_\mathcal{C} = \lambda X.\{a,b\} \cup Xa \cup Xb$, so that their Kleene iterates are $P^n_\mathcal{C}(\emptyset) = \{u \in \{a,b\}^* \mid |u| \leq n-1\}$ and $R^n_\mathcal{C}(\emptyset) = \{v \in \{a,b\}^+ \mid |v| \leq n\}$, for $n \in \mathbb{N}$. ⌟

A Finite Representation of $S_{\mathcal{A}}$. Given two vectors $X, X' \in \wp(\Sigma^*)^k$, we abuse notations and write $X \cup X'$ for the vector $\langle X_j \cup X'_j\rangle_{j\in[1,k]}$, and we write $X \subseteq X'$ when $X_j \subseteq X'_j$ for all $j \in [1, k]$. Given two functions $f : \wp(\Sigma^*)^k \to \wp(\Sigma^*)^k$ and $\rho : \wp(\Sigma^*) \to \wp(\Sigma^*)$ we write $f^n(\emptyset)$ for $f^n(\emptyset, \cdots, \emptyset)$, and $\rho \circ f$ for the function $\lambda X.\langle\rho((f(X))_j)\rangle_{j\in[1,k]}$.

Since $\leq$ is a wqo, $\rho_{\leq}(\operatorname{lfp} P_{\mathcal{A}}) = \rho_{\leq}(D)$ for some finite subset $D \subseteq_{\text{fin}} \operatorname{lfp} P_{\mathcal{A}}$. Since $\operatorname{lfp} P_{\mathcal{A}} = \bigcup_{n\in\mathbb{N}} P^n_{\mathcal{A}}(\emptyset)$, there exists some index $N_1 \in \mathbb{N}$ such that $D \subseteq P^{N_1}_{\mathcal{A}}(\emptyset)$. Hence, $\rho_{\leq}(P^{N_1}_{\mathcal{A}}(\emptyset)) = \rho_{\leq}(\operatorname{lfp} P_{\mathcal{A}})$ holds. This also applies to $\preccurlyeq$ and $R_{\mathcal{A},p}$, for each $p \in F$, so that there exists an index $N_2 \in \mathbb{N}$ such that $\rho_{\preccurlyeq}(R^{N_2}_{\mathcal{A},p}(\emptyset)) = \rho_{\preccurlyeq}(\operatorname{lfp} R_{\mathcal{A},p})$. Thus, by taking $T_p \triangleq P^{N_1}_{\mathcal{A}}(\emptyset) \times R^{N_2}_{\mathcal{A},p}(\emptyset)$, for each $p \in F$, we obtain a finite representation of $S_{\mathcal{A}}$, as required by step (2). By plugging the least fixpoint characterisation of $S_{\mathcal{A}}$ of Lemma 4.3 inside (1′), by observing that the closures preserve unions, and that $\rho_{\leq\times\preccurlyeq}$ and $\rho_{\leq} \times \rho_{\preccurlyeq}$ coincide on Cartesian products, we derive the following equivalences as in Section 2:

$$\begin{aligned} L^\omega(\mathcal{A}) \subseteq M &\iff \forall p \in F,\ \rho_{\leq\times\preccurlyeq}((\operatorname{lfp} P_{\mathcal{A}})_p \times (\operatorname{lfp} R_{\mathcal{A},p})_p) \subseteq I_M \\ &\iff \forall p \in F,\ \rho_{\leq\times\preccurlyeq}(T_p) \subseteq I_M \iff \forall p \in F,\ T_p \subseteq I_M \ . \end{aligned} \tag{2′}$$

▶ Remark 4.5. Assume that $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_n\}_{n\in\mathbb{N}}$ are two sequences of vectors in $\wp(\Sigma^*)^Q$ such that for each $n \in \mathbb{N}$: (i) the Q-indexed components of X_n and Y_n, for all n, are finite sets; (ii) for each $n \in \mathbb{N}$, $\rho_{\leq}(X_n) = \rho_{\leq}(P^n_{\mathcal{A}}(\emptyset))$ and $\rho_{\preccurlyeq}(Y_n) = \rho_{\preccurlyeq}(R^n_{\mathcal{A},p}(\emptyset))$. We call such a sequence $\{X_n\}_{n\in\mathbb{N}}$ (resp. $\{Y_n\}_{n\in\mathbb{N}}$) a sequence of *correct pruning steps* w.r.t. $\leq$ (resp. $\preccurlyeq$). Then, the vector $X_{N_1} \times Y_{N_2}$ can be used to achieve (2′), likewise T_p was used above. ⌟

Convergence Check. Let us now turn to the definition of a procedure for deciding when to stop the computations of $\rho_{\leq}(P^n_{\mathcal{A}}(\emptyset))$ and $\rho_{\preccurlyeq}(R^n_{\mathcal{A},p}(\emptyset))$. Here, we exploit a completeness property of the closures $\rho_{\leq}$ and $\rho_{\preccurlyeq}$, commonly used in abstract interpretation [8, 9]: a closure $\rho : C \to C$ is called *complete* for a function $f : C \to C$ when $\rho \circ f = \rho \circ f \circ \rho$ holds. Completeness is often used in abstract interpretation because it transfers to fixpoints, meaning that if ρ is complete for f then $\rho(\operatorname{lfp} f) = \operatorname{lfp}(\rho \circ f)$ holds [9, Theorem 7.1.0.4]. The following result provides a sufficient condition on a qo on Σ^* so as the induced closure operator turns out to be complete for the functions $P_{\mathcal{A}}$ and $R_{\mathcal{A},p}$, for each $p \in F$.

▶ **Lemma 4.6.** *Let $\mathcal{A} = (Q, \delta, i_{\mathcal{A}}, F)$ be a BA on Σ and $\leq$ be a right-monotonic qo on Σ^*. Then, $\rho_{\leq}$ is complete for $P_{\mathcal{A}}$ and $R_{\mathcal{A},p}$, for each $p \in F$.*

We are now in position to show that if the qos $\leq$ and $\preccurlyeq$ are right-monotonic and decidable, then a finite representation of $S_{\mathcal{A}}$ can be computed. First, observe that for all $n \geq 0$, $P^n_{\mathcal{A}}(\emptyset)$ is finite and computable (an easy induction can prove this). Let us also notice that $P_{\mathcal{A}}$ is a monotone function, hence $\rho_{\leq} \circ P_{\mathcal{A}}$ is monotone as well. Suppose that $\rho_{\leq}(P^{N_1+1}_{\mathcal{A}}(\emptyset)) \subseteq \rho_{\leq}(P^{N_1}_{\mathcal{A}}(\emptyset))$ holds for some $N_1 \in \mathbb{N}$. Thus, by monotonicity of $\rho_{\leq} \circ P_{\mathcal{A}}$, it turns out that $\rho_{\leq} \circ P_{\mathcal{A}} \circ \rho_{\leq}(P^{N_1+1}_{\mathcal{A}}(\emptyset)) \subseteq \rho_{\leq} \circ P_{\mathcal{A}} \circ \rho_{\leq}(P^{N_1}_{\mathcal{A}}(\emptyset))$. By Lemma 4.6, $\rho_{\leq}$ is complete for $P_{\mathcal{A}}$, hence this latter inclusion is equivalent to $\rho_{\leq}(P^{N_1+2}_{\mathcal{A}}(\emptyset)) \subseteq \rho_{\leq}(P^{N_1+1}_{\mathcal{A}}(\emptyset))$. A simple induction based on this argument proves that for all $k \geq N_1$, $\rho_{\leq}(P^k_{\mathcal{A}}(\emptyset)) \subseteq \rho_{\leq}(P^{N_1}_{\mathcal{A}}(\emptyset))$ holds, so that we obtain that $\{\rho_{\leq}(P^n_{\mathcal{A}}(\emptyset))\}_{n\in\mathbb{N}}$ finitely converges at iteration N_1. Hence, to detect convergence of the iterates we check whether $\rho_{\leq}(P^{n+1}_{\mathcal{A}}(\emptyset)) \subseteq \rho_{\leq}(P^n_{\mathcal{A}}(\emptyset))$ holds or not. When the qo $\leq$ is decidable, this test boils down to check if for each $x \in P^{n+1}_{\mathcal{A}}(\emptyset)$, there exists $y \in P^n_{\mathcal{A}}(\emptyset)$ such that $y \leq x$. This same reasoning also applies to $\preccurlyeq$ and $R_{\mathcal{A},p}$.

Word-based Inclusion Algorithms. Our "word-based" algorithm BAIncW for checking $L^\omega(\mathcal{A}) \subseteq M$ is parameterized by a pair of right-monotonic wqos $\leq, \preccurlyeq$ (on, resp., Σ^*, Σ^+) preserving I_M. It computes the Kleene iterates $P^n_{\mathcal{A}}(\emptyset)$ and $R^n_{\mathcal{A},p}(\emptyset)$, for each final state

$p \in F$, until $\rho_{\leq}((P_{\mathcal{A}}^{N_1+1}(\emptyset))_q) \subseteq \rho_{\leq}((P_{\mathcal{A}}^{N_1}(\emptyset))_q)$ and $\rho_{\preccurlyeq}((R_{\mathcal{A},p}^{N_2+1}(\emptyset))_q) \subseteq \rho_{\preccurlyeq}((R_{\mathcal{A},p}^{N_2}(\emptyset))_q)$ hold for each $q \in Q$ and some $N_1, N_2 \in \mathbb{N}$. The resulting finite sets of words $(P_{\mathcal{A}}^{N_1}(\emptyset))_p$ and $(R_{\mathcal{A},p}^{N_2}(\emptyset))_p$, for each final state $p \in F$, are used by the membership check procedure enabled by (2′):

$$L^\omega(\mathcal{A}) \subseteq M \iff \forall p \in F, \forall u \in (P_{\mathcal{A}}^{N_1}(\emptyset))_p, \forall v \in (R_{\mathcal{A},p}^{N_2}(\emptyset))_p,\ uv^\omega \in M.$$

BAIncW Word-based algorithm for checking $L^\omega(\mathcal{A}) \subseteq M$.

```
  Data: Büchi automaton $\mathcal{A} = (Q, \delta, i_{\mathcal{A}}, F)$
  Data: Procedure deciding $uv^\omega \overset{?}{\in} M$ given $(u, v) \in \Sigma^* \times \Sigma^+$
  Data: Decidable right-monotonic wqos $\leq, \preccurlyeq$ s.t. $\rho_{\leq \times \preccurlyeq}(I_M) = I_M$
1 Compute $P_{\mathcal{A}}^{N_1}(\emptyset)$ with least $N_1$ s.t. $\forall q \in Q$, $\rho_{\leq}((P_{\mathcal{A}}^{N_1+1}(\emptyset))_q) \subseteq \rho_{\leq}((P_{\mathcal{A}}^{N_1}(\emptyset))_q)$;
2 foreach $p \in F$ do
3     Compute $R_{\mathcal{A},p}^{N_2}(\emptyset)$ with least $N_2$ s.t. $\forall q \in Q$, $\rho_{\preccurlyeq}((R_{\mathcal{A},p}^{N_2+1}(\emptyset))_q) \subseteq \rho_{\preccurlyeq}((R_{\mathcal{A},p}^{N_2}(\emptyset))_q)$;
4     foreach $u \in (P_{\mathcal{A}}^{N_1}(\emptyset))_p$, $v \in (R_{\mathcal{A},p}^{N_2}(\emptyset))_p$ do
5         if $uv^\omega \notin M$ then return false;
6 return true;
```

▶ **Theorem 4.7.** *Given all the required input data,* BAIncW *decides* $L^\omega(\mathcal{A}) \subseteq M$*.*

▶ Remark 4.8. The for-loop at lines 2-5 of BAIncW is restricted to the final states $p \in F$ of the BA $\mathcal{A}$. Thus, in general, the less they are the better is for BAIncW. ⌟

▶ **Example 4.9.** Consider the BAs $\mathcal{C}$ and $\mathcal{D}$ in Fig. 1. From Example 4.4 we have that $P_{\mathcal{C}}(\emptyset) = \{\epsilon\}$, $P_{\mathcal{C}}^2(\emptyset) = \{\epsilon, a, b\}$ and $P_{\mathcal{C}}^3(\emptyset) = \{\epsilon, a, b, aa, ab, ba, bb\}$. From Example 3.1, for $u \in \{aa, ba\}$ and $v \in \{ab, bb\}$, we have that $a \leq^{\mathcal{D}} u$ and $b \leq^{\mathcal{D}} v$, while a and ϵ are incomparable for $\leq^{\mathcal{D}}$. Hence, $\rho_{\leq^{\mathcal{D}}}(P_{\mathcal{C}}(\emptyset)) \neq \rho_{\leq^{\mathcal{D}}}(P_{\mathcal{C}}^2(\emptyset))$ and $\rho_{\leq^{\mathcal{D}}}(P_{\mathcal{C}}^2(\emptyset)) = \rho_{\leq^{\mathcal{D}}}(P_{\mathcal{C}}^3(\emptyset))$ hold, so that a finite representation of lfp $P_{\mathcal{C}}$ is achieved by $P_{\mathcal{C}}^2(\emptyset)$. Since $\rho_{\preccurlyeq^{\mathcal{D}}}(R_{\mathcal{C}}^2(\emptyset)) = \rho_{\preccurlyeq^{\mathcal{D}}}(R_{\mathcal{C}}^1(\emptyset))$, the membership check is performed on the elements of $P_{\mathcal{C}}^2(\emptyset) \times R_{\mathcal{C}}^1(\emptyset) = \{\epsilon, a, b\} \times \{a, b\}$, and for $(a, b) \in P_{\mathcal{C}}^2(\emptyset) \times R_{\mathcal{C}}^1(\emptyset)$, the word ab^ω is a witness that $L^\omega(\mathcal{C}) \not\subseteq L^\omega(\mathcal{D})$. ⌟

As explained by Remark 4.5, any sequence of correct pruning steps for the Kleene iterates can be safely exploited to compute a finite representation of $S_{\mathcal{A}}$. This is formalized by the algorithm BAIncW given in App. A.

The pairs of qos derived from M as defined in Section 3, are all pairs of decidable right-monotonic wqos that verify the preservation property w.r.t. M. Each of them yields a slightly different algorithm deciding whether $L^\omega(\mathcal{A}) \subseteq M$ holds (see the discussion in Section 4.3).

4.2 Language Inclusion ω-context-free $\subseteq$ ω-regular

A (*Büchi*) *pushdown automaton* ((B)PDA) on Σ is a tuple $\mathcal{P} = (Q, \Gamma, \delta, i, F)$ where Q is a finite set of states including an initial state i, Γ is the stack alphabet including an initial stack symbol $\bot$, $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \times Q \times \Gamma^*$ is the finite set of transitions, and $F \subseteq Q$ is a subset of accepting states. Configurations of the PDA $\mathcal{P}$ are pairs in $Q \times \Gamma^*$ and, for each $a \in \Sigma$, the transition relation $\vdash^a$ between configurations is defined by $(q, \gamma w) \vdash^a (p, \beta w)$, for some $w \in \Gamma^*$, when $(q, a, \gamma, p, \beta) \in \delta$, and it is lifted to words by reflexivity and transitivity, that is, for all $u \in \Sigma^*$, $(q, w) \vdash^{*u} (p, w')$ when the configurations (q, w) and (p, w') are related by a sequence of transitions such that the concatenation of the corresponding labels is the word u. We write $(q, w) \vdash^{*u}_F (p, w')$ when such a sequence includes a configuration whose state is final. The language of finite words accepted by a

PDA $\mathcal{P}$ is $L^*(\mathcal{P}) \triangleq \{u \in \Sigma^* \mid (i, \perp) \vdash^{*u} (p, w),\ p \in F,\ w \in \Gamma^*\}$. A natural extension from finite to infinite words relies on infinite sequences of configurations as follows. A trace of $\mathcal{P}$ for an ω-word $\xi = a_0 a_1 \cdots \in \Sigma^\omega$ is an infinite sequence $(q_0, w_0) \vdash^{*a_0} (q_1, w_1) \vdash^{*a_1} \cdots$, which is initial when $(q_0, w_0) = (i, \perp)$ and fair when $q_j \in F$ for infinitely many j's. The ω-language accepted by $\mathcal{P}$ is $L^\omega(\mathcal{P}) \triangleq \{\xi \in \Sigma^\omega \mid$ there exists an initial and fair trace of $\mathcal{P}$ for $\xi\}$. An ω-language $L \subseteq \Sigma^\omega$ is ω-context-free if $L = L^\omega(\mathcal{P})$ for some BPDA $\mathcal{P}$ on Σ.

We fix an ω-regular language M, a pair $\leq, \preccurlyeq$ of monotonic wqos on Σ^*, Σ^+ such that $\rho_{\leq\times\preccurlyeq}(I_M) = I_M$ holds, and a BPDA $\mathcal{P}$ such that $L = L^\omega(\mathcal{P})$. Theorem 4.1 still holds when the "left" language L is ω-context-free, so that $L \subseteq M \iff I_L \subseteq I_M$ holds. The following result generalises Lemma 4.2 to BPDAs.

▶ **Lemma 4.10.** *Let $\mathcal{P} = (Q, \Gamma, \delta, i, F)$ be a BPDA. Then, $uv^\omega \in L^\omega(\mathcal{P})$ iff there exist $(q, \gamma) \in Q \times \Gamma$, $u' \in \Sigma^*$, $v' \in \Sigma^+$ such that $uv^\omega = u'v'^\omega$, $(i, \perp) \vdash^{*u'} (q, \gamma s)$ and $(q, \gamma) \vdash^{*v'}_{F} (q, \gamma w)$, for some $s, w \in \Gamma^*$.*

Similarly to the ω-regular case described in Section 4.1, Lemma 4.10 allows us to define two PDAs $\mathcal{P}^1_{q\gamma}$ and $\mathcal{P}^2_{q\gamma}$, where for each $(q, \gamma) \in Q \times \Gamma$, $\mathcal{P}^1_{q\gamma}$ deals with the prefixes, $\mathcal{P}^2_{q\gamma}$ deals with the periods, and are such that the ultimately periodic words generated by the pairs in $S_{\mathcal{P}} \triangleq \bigcup_{(q,\gamma)\in Q\times\Gamma} L^*(\mathcal{P}^1_{q\gamma}) \times L^*(\mathcal{P}^2_{q\gamma})$ coincide with those of $L^\omega(\mathcal{P})$. Hence, similarly to (1′) for the ω-regular case, it turns out that:

$$L^\omega(\mathcal{P}) \subseteq M \iff S_{\mathcal{P}} \subseteq I_M \iff \rho_{\leq\times\preccurlyeq}(S_{\mathcal{P}}) \subseteq I_M \ . \tag{1''}$$

Moreover, analogously to Lemma 4.3 for the ω-regular case, $S_{\mathcal{P}}$ admits a least fixpoint characterisation.

▶ **Lemma 4.11.** *Any PDA $\mathcal{P}$ induces a monotone map $F_{\mathcal{P}} : \wp(\Sigma^*)^m \to \wp(\Sigma^*)^m$, for some $m \in \mathbb{N}$, such that $L^*(\mathcal{P}) = (\operatorname{lfp} F_{\mathcal{P}})_0$.*

Let us mention that the definition of $F_{\mathcal{P}}$ relies on the production rules of a context-free grammar (CFG) accepting $L^*(\mathcal{P})$ and that $(\operatorname{lfp} F_{\mathcal{P}})_0$ denotes the first vector component corresponding to the start variable of the CFG. Let $P_{q\gamma}$ and $R_{q\gamma}$ be the functions provided by Lemma 4.11 for the two PDAs $\mathcal{P}^1_{q\gamma}$ and $\mathcal{P}^2_{q\gamma}$ defined above for each $(q, \gamma) \in Q \times \Gamma$. By (1″) and Lemma 4.11, it turns out that:

$$L^\omega(\mathcal{P}) \subseteq M \iff \forall (q, \gamma) \in Q \times \Gamma,\ \rho_{\leq}((\operatorname{lfp} P_{q\gamma})_0) \times \rho_{\preccurlyeq}((\operatorname{lfp} R_{q\gamma})_0)) \subseteq I_M \ . \tag{2''}$$

Since both $\leq$ and $\preccurlyeq$ are wqos, the corresponding upward-closed sets in (2″) can be obtained as upward closure of some finite subsets. In particular, by reasoning as for the ω-regular case, we have that for each $(q, \gamma) \in Q \times \Gamma$ there exist $N_1, N_2 \in \mathbb{N}$ such that $\rho_{\leq}((\operatorname{lfp} P_{q\gamma})_0) = \rho_{\leq}((P^{N_1}_{q\gamma}(\emptyset))_0)$ and $\rho_{\preccurlyeq}((\operatorname{lfp} R_{q\gamma})_0) = \rho_{\preccurlyeq}((R^{N_2}_{q\gamma}(\emptyset))_0)$ hold.

Let us now turn to the convergence of the sequences of Kleene iterates. Being induced by the rules of a CFG, the function $F_{\mathcal{P}}(\langle X_1, ..., X_m\rangle)$ of Lemma 4.11 may rely on nonlinear concatenations of type $X_i X_j$ for some $i, j \in [1, m]$, so that prefixes and periods in $S_{\mathcal{P}}$ can be obtained both by left- and right-concatenations. This is different from the ω-regular case, where only right-concatenations were needed. Thus, in contrast to the ω-regular case of Lemma 4.6, we need stronger monotonicity conditions on the qos $\leq$ and $\preccurlyeq$ in order to ensure the completeness of the closures $\rho_{\leq}$ and $\rho_{\preccurlyeq}$ for, resp., $P_{q\gamma}$ and $R_{q\gamma}$: both qos need to be (left- and right-) monotonic.

▶ **Lemma 4.12.** *Let $\mathcal{P}$ be a BPDA on Σ and $\leq$ be a monotonic qo on Σ^*. Then, $\rho_{\leq}$ is complete for all the functions $P_{q\gamma}$ and $R_{q\gamma}$ induced by $\mathcal{P}$.*

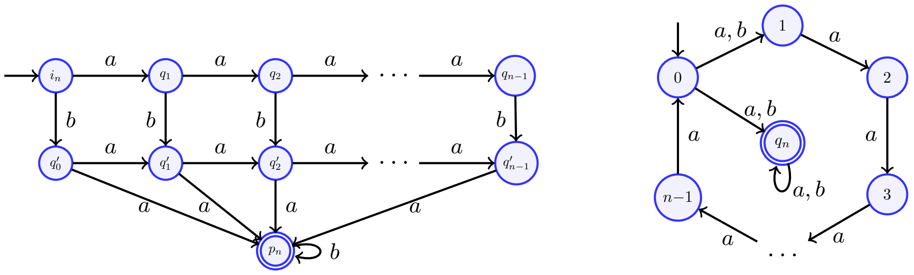


Figure 2 The families $\{\mathcal{A}_n\}_{n\geq 2}$ (left) and $\{\mathcal{B}_n\}_{n\geq 2}$ (right) s.t. $L^\omega(\mathcal{A}_n) \subseteq L^\omega(\mathcal{B}_n)$ for all n.

Hence, by Lemma 4.12, the same arguments used for the ω-regular case entail that the convergence of the Kleene iterates of $P_{q\gamma}$ and $R_{q\gamma}$ boils down to check, resp., the conditions: $\rho_{\leq}(P^{n+1}_{q\gamma}(\emptyset)) \subseteq \rho_{\leq}(P^{n}_{q\gamma}(\emptyset))$ and $\rho_{\preccurlyeq}(R^{n+1}_{q\gamma}(\emptyset)) \subseteq \rho_{\preccurlyeq}(R^{n}_{q\gamma}(\emptyset))$, for some $n \in \mathbb{N}$.

Summing up, our "word-based" algorithm for $L^\omega(\mathcal{P}) \subseteq M$ follows the same template of BAIncW for the ω-regular case. First, it computes the iterates of $P_{q\gamma}$ and $R_{q\gamma}$ for, resp., the prefix and period languages, until finite convergence is reached. Then, the resulting finite sets of words $(P^{N_1}_{q\gamma}(\emptyset))_0$ and $(R^{N_2}_{q\gamma}(\emptyset))_0$ are used by the following membership check:

$$L^\omega(\mathcal{P}) \subseteq M \iff \forall (q,\gamma) \in Q \times \Gamma, \forall u \in (P^{N_1}_{q\gamma}(\emptyset))_0, \forall v \in (R^{N_1}_{q\gamma}(\emptyset))_0,\ uv^\omega \in M \ .$$

The pairs of state-based wqos that can be used to decide the inclusion $L^\omega(\mathcal{P}) \subseteq M$ are $\trianglelefteq^{\mathcal{B}}, \preccurlyeq^{\mathcal{B}}$ and $\preccurlyeq^{\mathcal{B}}, \preccurlyeq^{\mathcal{B}}$, where $\mathcal{B}$ is a BA recognising M, as defined in Section 3.

4.3 Discussion

Let us discuss how the inclusion algorithms provided by pairs of qos defined in Section 3 can be related to each other. Consider two wqos $\leq, \leq' \subseteq \Sigma^* \times \Sigma^+$ such that $\leq$ is coarser than $\leq'$, i.e., $\leq' \subseteq \leq$ holds. It turns out that $\rho_{\leq'}(X) \subseteq \rho_{\leq'}(Y)$ implies $\rho_{\leq}(X) \subseteq \rho_{\leq}(Y)$, so that if some Kleene iterates of BAIncW converge in N' steps w.r.t. $\leq'$, then the same Kleene iterates converge in $N \leq N'$ steps w.r.t. $\leq$, namely, convergence can be "faster" with a coarser qo. Also, given a wqo $\leq$ and a nonempty set $X \in \wp(\Sigma^*)$, consider the set $\mathcal{C}_X \triangleq \{Y \subseteq_{\text{fin}} X \mid \rho_{\leq}(Y) = \rho_{\leq}(X)\}$ of finite subsets of X inducing the same $\leq$-upward closure as X, which is not empty because $\leq$ is a wqo. An element of $\mathcal{C}_X$ of minimal size is called a *minor* of X and denoted by $\lfloor X \rfloor_{\leq}$. If $\leq$ is coarser than $\leq'$ then any minor $\lfloor X \rfloor_{\leq}$ w.r.t. $\leq$ has at most as many elements as any minor $\lfloor X \rfloor_{\leq'}$ w.r.t. $\leq'$. Thus, a coarser pair of wqos may achieve a smaller minimal representation on which to perform the membership queries of BAIncW. The following example shows the benefits of using the coarsest state-based pair of wqos on the family of inclusion problems between the BAs depicted in Fig. 2.

▶ **Example 4.13.** Consider the families of BAs $\{\mathcal{A}_n\}_{n\geq 2}$ and $\{\mathcal{B}_n\}_{n\geq 2}$ in Fig. 2. Let $X_n \triangleq \{a^iba^{j+1} \in \Sigma^* \mid i,j \geq 0,\ i{+}j \leq n{-}1\}$ such that $L^*(\mathcal{A}_n{}^{i_n}_{p_n}) = X_n\{b\}^*$ and $L^*(\mathcal{A}_n{}^{p_n}_{p_n})\backslash\{\epsilon\} = b^+$. For any $w \in L^*(\mathcal{A}_n{}^{i_n}_{p_n})$ we have that $q_n \in s^{\mathcal{B}_n}[w]$, and, since $s^{\mathcal{B}_n}[aba] = \{q_n\}$, it holds that $aba \leq^{\mathcal{B}_n} w$. Since $aba \in L^*(\mathcal{A}_n{}^{i_n}_{p_n})$, we deduce that any minor $\lfloor L^*(\mathcal{A}_n{}^{i_n}_{p_n})\rfloor_{\leq^{\mathcal{B}_n}}$ has size one. Similarly, any minor $\lfloor L^*(\mathcal{A}_n{}^{p_n}_{p_n})\backslash\{\epsilon\}\rfloor_{\preccurlyeq^{\mathcal{B}_n}}$ has size one. We also have that $c^{\mathcal{B}_n}[a^iba^{j+1}] = \{(n-i, j+2), (0, q_n), (q_n, q_n)\}$. Hence, if $w \preccurlyeq^{\mathcal{B}_n} w'$, for $w, w' \in X_n$, then $w = w'$. Since X_n has size $\frac{n(n+1)}{2}$, all the minors $\lfloor L^*(\mathcal{A}_n{}^{i_n}_{p_n})\backslash\{\epsilon\}\rfloor_{\preccurlyeq^{\mathcal{B}_n}}$ and $\lfloor L^*(\mathcal{A}_n{}^{i_n}_{p_n})\backslash\{\epsilon\}\rfloor_{\trianglelefteq^{\mathcal{B}_n}}$ have at least $\frac{n(n+1)}{2}$ elements. Hence, using the pair of qos $\leq^{\mathcal{B}_n}, \preccurlyeq^{\mathcal{B}_n}$, a single membership query (i.e., $uv^\omega \in L^\omega(\mathcal{B}_n)$) is needed to decide the inclusion $L^\omega(\mathcal{A}_n) \subseteq L^\omega(\mathcal{B}_n)$, as opposed to no less than $\frac{n(n+1)}{2}$ membership queries for the other pairs of qos. ⌟

▶ Remark 4.14. The supergraphs of [2, Def. 6] endowed with their subsumption orders coincide with our qo $\preccurlyeq$. Without the subsumption order they coincide with $\preccurlyeq \cap \preccurlyeq^{-1}$. ⌟

5 State-Based Inclusion Algorithms

In this section, we show how to derive *state-based* inclusion algorithms, namely, algorithms that, given two BAs $\mathcal{A} = (Q_\mathcal{A}, \delta_\mathcal{A}, i_\mathcal{A}, F_\mathcal{A})$ and $\mathcal{B} = (Q_\mathcal{B}, \delta_\mathcal{B}, i_\mathcal{B}, F_\mathcal{B})$, decide whether $L^\omega(\mathcal{A}) \subseteq L^\omega(\mathcal{B})$ by operating on the states of $\mathcal{A}$ and $\mathcal{B}$ only. The intuition is that words are abstracted into states and, correspondingly, operations/tests on words are abstracted into operations/tests on states. Of course, the key to enable such abstractions are the state-based qos defined in Section 3, whose definitions rely just on the states of a BA representing an ω-language. Due to lack of space, we focus on the ω-regular case, while a state-based algorithm for the context-free case is given in App. B and is designed by following an analogous pattern.

We focus on the pair of qos $\leq^\mathcal{B}, \preccurlyeq^\mathcal{B}$ defined in Section 3. The state-based algorithms for different pairs of qos can be analogously derived. Given an ultimately periodic word uv^ω, the prefix $u \in \Sigma^*$ is abstracted by the set of its successor states in $\mathcal{B}$ given by $s^\mathcal{B}[u] \in \wp(Q_\mathcal{B})$, while the period v is abstracted by the pair $(c^\mathcal{B}[v], f^\mathcal{B}[v]) \in \wp(Q_\mathcal{B}^2) \times \wp(Q_\mathcal{B}^2)$ providing its context and final context in $\mathcal{B}$. Thus, the state abstraction of $S_\mathcal{A}$, as given in Lemma 4.3, is:

$$S_{\mathcal{A},\mathcal{B}} \triangleq \textstyle\bigcup_{p\in F_\mathcal{A}} \{s^\mathcal{B}[u] \mid u \in (\operatorname{lfp} P_\mathcal{A})_p\} \times \{(c^\mathcal{B}[v], f^\mathcal{B}[v]) \mid v \in (\operatorname{lfp} R_{\mathcal{A},p})_p\} \ .$$

We give a fixpoint characterisation of $S_{\mathcal{A},\mathcal{B}}$ using the state abstractions of the functions $P_\mathcal{A}$ and $R_{\mathcal{A},p}$ w.r.t., resp., the qos $\leq^\mathcal{B}$ and $\preccurlyeq^\mathcal{B}$.

Let us define the maps $\operatorname{Post}_\mathcal{A}^{\leq^\mathcal{B}} : \wp(\wp(Q_\mathcal{B}))^{Q_\mathcal{A}} \to \wp(\wp(Q_\mathcal{B}))^{Q_\mathcal{A}}$ and $\operatorname{Post}_\mathcal{A}^{\preccurlyeq^\mathcal{B}} : \wp(\wp(Q_\mathcal{B}^2) \times \wp(Q_\mathcal{B}^2))^{Q_\mathcal{A}} \to \wp(\wp(Q_\mathcal{B}^2) \times \wp(Q_\mathcal{B}^2))^{Q_\mathcal{A}}$ as follows:

$$\operatorname{Post}_\mathcal{A}^{\leq^\mathcal{B}}(X) \triangleq \langle \textstyle\bigcup_{a\in\Sigma, q\in\delta_\mathcal{A}(q',a)} \{y \star a \mid y \in X_{q'}\}\rangle_{q\in Q_\mathcal{A}}$$
$$\operatorname{Post}_\mathcal{A}^{\preccurlyeq^\mathcal{B}}(Y) \triangleq \langle \textstyle\bigcup_{a\in\Sigma, q\in\delta_\mathcal{A}(q',a)} \{\big(y_1 \circ c^\mathcal{B}[a], y_1 \circ f^\mathcal{B}[a] \cup y_2 \circ c^\mathcal{B}[a]\big) \mid (y_1, y_2) \in Y_{q'}\}\rangle_{q\in Q_\mathcal{A}}$$

where $y \star a \triangleq \bigcup_{q'\in y} \{q \in Q_\mathcal{B} \mid (q', q) \in c^\mathcal{B}[a]\}$, for $y \in \wp(Q_\mathcal{B})$ and $a \in \Sigma$. The intuition for this latter definition is the following: if $y = s^\mathcal{B}[u]$, for some $u \in \Sigma^*$, then $y \star a = s^\mathcal{B}[ua]$. Also, given two binary relations $y_1, y_2 \in \wp(Q_\mathcal{B}^2)$ on states of $\mathcal{B}$, the notation $y_1 \circ y_2$ denotes their composition. Here, the intuition is similar: if $y_1 = c^\mathcal{B}[u]$ and $y_2 = f^\mathcal{B}[u]$, for some $u \in \Sigma^*$, then $y_1 \circ c^\mathcal{B}[a] = c^\mathcal{B}[ua]$ and $y_1 \circ f^\mathcal{B}[a] \cup y_2 \circ c^\mathcal{B}[a] = f^\mathcal{B}[ua]$. In turn, the functions:

$$P_{\mathcal{A},\mathcal{B}} \triangleq \lambda X \in \wp(\wp(Q_\mathcal{B}))^{Q_\mathcal{A}}.\ \langle\{\{i_\mathcal{B}\} \mid q = i_\mathcal{A}\}\rangle_{q\in Q_\mathcal{A}} \cup \operatorname{Post}_\mathcal{A}^{\leq^\mathcal{B}}(X)$$
$$R_{\mathcal{A},\mathcal{B},p} \triangleq \lambda Y \in \wp(\wp(Q_\mathcal{B}^2) \times \wp(Q_\mathcal{B}^2))^{Q_\mathcal{A}}.\ \langle\{(c^\mathcal{B}[a], f^\mathcal{B}[a]) \mid q \in \delta_\mathcal{A}(p, a)\}\rangle_{q\in Q_\mathcal{A}} \cup \operatorname{Post}_\mathcal{A}^{\preccurlyeq^\mathcal{B}}(Y)$$

with $p \in Q_\mathcal{A}$, give us the following least fixpoint characterization:

▶ **Lemma 5.1.** $S_{\mathcal{A},\mathcal{B}} = \bigcup_{p\in F_\mathcal{A}} (\operatorname{lfp} P_{\mathcal{A},\mathcal{B}})_p \times (\operatorname{lfp} R_{\mathcal{A},\mathcal{B},p})_p$.

Let us now turn to the convergence check for the Kleene iterates of $P_{\mathcal{A},\mathcal{B}}$ and $R_{\mathcal{A},\mathcal{B},p}$. The qo $\leq^\mathcal{B}$ on words translates into the inclusion order $\subseteq$ on $\wp(Q_\mathcal{B})$ and, analogously, $\preccurlyeq^\mathcal{B}$ translates into the componentwise inclusion order $\subseteq^2 \triangleq \subseteq \times \subseteq$ on $\wp(Q_\mathcal{B}^2) \times \wp(Q_\mathcal{B}^2)$. Hence, the convergence of the iterates $P_{\mathcal{A},\mathcal{B}}^n(\emptyset)$ is checked by $\rho_\subseteq(P_{\mathcal{A},\mathcal{B}}^{n+1}(\emptyset)) \subseteq \rho_\subseteq(P_{\mathcal{A},\mathcal{B}}^n(\emptyset))$ (where $\subseteq$ is componentwise on vectors). Similarly, for the iterates $R_{\mathcal{A},\mathcal{B},p}^n(\emptyset)$ w.r.t. $\subseteq^2$. Let us remark that since the inclusion $\subseteq$ is a partial order (rather than a mere qo), each set $X \in \wp(\wp(Q_\mathcal{B}))$ admits a unique minor $\lfloor X \rfloor$ w.r.t. $\subseteq$, and similarly for $\subseteq^2$. Hence, the sequences of minors $\{\lfloor P_{\mathcal{A},\mathcal{B}}^n(\emptyset) \rfloor\}_{n\in\mathbb{N}}$ and $\{\lfloor R_{\mathcal{A},\mathcal{B},p}^n(\emptyset) \rfloor\}_{n\in\mathbb{N}}$ w.r.t., resp., $\subseteq$ and $\subseteq^2$, are uniquely defined. Since these are sequences of correct pruning steps according to Remark 4.5, they can be exploited to achieve a smaller representation of $S_{\mathcal{A},\mathcal{B}}$. Hence, the clear rationale to use these uniquely defined minors is to keep at each iteration the minimum number of elements of the Kleene iterates for representing them.

Finally, let us discuss the state abstraction of the membership check $uv^\omega \in L^\omega(\mathcal{B})$. For $x \in \wp(Q_\mathcal{B})$ and $(y_1, y_2) \in \wp(Q^2_\mathcal{B}) \times \wp(Q^2_\mathcal{B})$, define the following state-based inclusion predicate:

$$\mathrm{Inc}^\mathcal{B}(x, (y_1, y_2)) \triangleq \exists q, q' \in Q_\mathcal{B},\ q \in x \wedge (q, q') \in y_1^* \wedge (q', q') \in y_1^* \circ y_2 \circ y_1^* \ .$$

This is the correct state-based membership check because for all $u \in \Sigma^*$, $v \in \Sigma^+$, it turns out that $uv^\omega \in L^\omega(\mathcal{B}) \Leftrightarrow \mathrm{Inc}^\mathcal{B}(s^\mathcal{B}[u], (c^\mathcal{B}[v], f^\mathcal{B}[v]))$.

Summing up, we are now in a position to put forward our state-based algorithm BAIncS for checking $L^\omega(\mathcal{A}) \subseteq L^\omega(\mathcal{B})$. An illustrative run on the example of Fig. 1 is given in Section 5.1.

BAIncS State-based algorithm for checking $L^\omega(\mathcal{A}) \subseteq L^\omega(\mathcal{B})$.

```
Data: Büchi automata $\mathcal{A} = (Q_\mathcal{A}, \delta_\mathcal{A}, i_\mathcal{A}, F_\mathcal{A})$ and $\mathcal{B} = (Q_\mathcal{B}, \delta_\mathcal{B}, i_\mathcal{B}, F_\mathcal{B})$
Compute $\lfloor P^{N_1}_{\mathcal{A},\mathcal{B}}(\emptyset) \rfloor$ with least $N_1$ s.t. $\forall q \in Q_\mathcal{A}$, $\rho_\subseteq((P^{N_1+1}_{\mathcal{A},\mathcal{B}}(\emptyset))_q) \subseteq \rho_\subseteq((P^{N_1}_{\mathcal{A},\mathcal{B}}(\emptyset))_q)$;
foreach $p \in F_\mathcal{A}$ do
  Compute $\lfloor R^{N_2}_{\mathcal{A},\mathcal{B},p}(\emptyset) \rfloor$ with least $N_2$ s.t. $\forall q \in Q_\mathcal{A}$, $\rho_{\subseteq^2}((R^{N_2+1}_{\mathcal{A},\mathcal{B},p}(\emptyset))_q) \subseteq \rho_{\subseteq^2}((R^{N_2}_{\mathcal{A},\mathcal{B},p}(\emptyset))_q)$;
  foreach $x \in (\lfloor P^{N_1}_{\mathcal{A},\mathcal{B}}(\emptyset) \rfloor)_p$, $(y_1, y_2) \in (\lfloor R^{N_2}_{\mathcal{A},\mathcal{B},p}(\emptyset) \rfloor)_p$ do
      if $\neg\mathrm{Inc}^\mathcal{B}(x, (y_1, y_2))$ then return false;
return true;
```

▶ **Theorem 5.2.** *The algorithm* BAIncS *decides* $L^\omega(\mathcal{A}) \subseteq L^\omega(\mathcal{B})$.

5.1 Illustrative Example of BAIncS

We show the execution of a run of BAIncS on the BAs $\mathcal{C}$ and $\mathcal{D}$ depicted in Fig. 1. As a result, the algorithm will correctly decide that $L^\omega(\mathcal{C})$ is not included in $L^\omega(\mathcal{D})$ (e.g., $ab^\omega \in L^\omega(\mathcal{C})$ but $ab^\omega \notin L^\omega(\mathcal{D})$). Observe that since $\mathcal{C}$ consists of a single state, vectors are not needed.

First, the algorithm evaluates the sequence $\{\lfloor P^n_{\mathcal{C},\mathcal{D}}(\emptyset) \rfloor\}_{n \in \mathbb{N}} \in (\wp(\wp(Q_\mathcal{D})))^\mathbb{N}$, where $P_{\mathcal{C},\mathcal{D}}(X) = \{\{q_0\}\} \cup \{x \star a \mid x \in X\} \cup \{x \star b \mid x \in X\}$.

(1) $\lfloor P^1_{\mathcal{C},\mathcal{D}}(\emptyset) \rfloor = \{\{q_0\}\}$,

(2) $\lfloor P^2_{\mathcal{C},\mathcal{D}}(\emptyset) \rfloor = \lfloor \{\{q_0\}\} \cup \{\{q_0\} \star a, \{q_0\} \star b\} \rfloor = \{\{q_0\}, \{q\}\}$,

(3) $\lfloor P^3_{\mathcal{C},\mathcal{D}}(\emptyset) \rfloor = \lfloor \{\{q_0\}\} \cup \{\{q_0\} \star a, \{q_0\} \star b, \{q\} \star a, \{q\} \star b\} \rfloor = \{\{q_0\}, \{q\}\}$.

Hence, $\lfloor P^3_{\mathcal{C},\mathcal{D}}(\emptyset) \rfloor = \lfloor P^2_{\mathcal{C},\mathcal{D}}(\emptyset) \rfloor$ and the computations for the prefix iterates stop at the third iteration.

Next, the algorithm evaluates the sequence $\{\lfloor R^n_{\mathcal{C},\mathcal{D}}(\emptyset) \rfloor\}_{n \in \mathbb{N}} \in (\wp(\wp(Q^2_\mathcal{D}) \times \wp(Q^2_\mathcal{D})))^\mathbb{N}$. Let $y \triangleq \{(q_0, q), (q, q)\}$, $z_1 \triangleq \{(q_0, q_0), (q, q_0)\}$ and $z_2 \triangleq \{(q, q_0)\}$. We have that $y, z_1, z_2 \in \wp(Q^2_\mathcal{D})$, $y = c^\mathcal{D}[a] = f^\mathcal{D}[a]$, $z_1 = c^\mathcal{D}[b]$, $z_2 = f^\mathcal{D}[b]$ and $\{(c^\mathcal{D}(c), f^\mathcal{D}(c)) \mid i \xrightarrow{c} i \wedge c \in \{a, b\}\} = \{(y, y), (z_1, z_2)\}$. For each pair $p = (p_1, p_2) \in \wp(Q^2_\mathcal{D}) \times \wp(Q^2_\mathcal{D})$ and each $c \in \Sigma^*$, we define $p * c \triangleq p_1 \circ f^\mathcal{D}(c) \cup p_2 \circ c^\mathcal{D}(c) \in \wp(Q^2_\mathcal{D})$. We then have:

(1) $R^1_{\mathcal{C},\mathcal{D}}(X) = \{(y, y), (z_1, z_2)\} \cup \mathrm{Post}^{\preccurlyeq^\mathcal{D}}_\mathcal{C}(X)$

$$= \{(y, y), (z_1, z_2)\} \cup \{(p_1 \circ c^\mathcal{D}[c], p * c) \mid i \xrightarrow{c} i \wedge c \in \{a, b\} \wedge (p_1, p_2) \in X\},$$

so that $\lfloor R^1_{\mathcal{C},\mathcal{D}}(\emptyset) \rfloor = \lfloor \{(y, y), (z_1, z_2)\} \rfloor = \{(y, y), (z_1, z_2)\}$.

(2) $\lfloor R^2_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor =$

$$\lfloor\{(y,y),(z_1,z_2)\} \cup \{(y \circ c^{\mathcal{D}}[a],(y,y)*a),(z_1 \circ c^{\mathcal{D}}[a],(z_1,z_2)*a),$$
$$(y \circ c^{\mathcal{D}}[b],(y,y)*b),(z_1 \circ c^{\mathcal{D}}[b],(z_1,z_2)*b)\}\rfloor =$$
$$\lfloor\{(y,y),(z_1,z_2)\} \cup \{(y,y),(z_1,z_2),(z_1,z_1)\}\rfloor.$$

Since $(z_1,z_2) \sqsubseteq^2 (z_1,z_1)$, $\lfloor R^2_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor = \lfloor\{(y,y),(z_1,z_2),(z_1,z_1)\}\rfloor = \{(y,y),(z_1,z_2)\}$.

Thus, $\lfloor R^2_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor = \lfloor R^1_{\mathcal{C},\mathcal{D}}(\emptyset)\rfloor$ and the computations for the period iterates stop at the second iteration.

It turns out that $\neg\mathrm{Inc}^{\mathcal{D}}(\{q\},(z_1,z_2))$: this, intuitively, corresponds to the counterexample ab^ω that belongs to $L^\omega(\mathcal{C})$ but not $L^\omega(\mathcal{D})$. Hence, the inclusion $L^\omega(\mathcal{C}) \subseteq L^\omega(\mathcal{D})$ does not hold.

6 Implementation and Experimental Evaluation

Benchmarks. We collected new benchmarks from various trusted sources that significantly expand the set of problem instances available to the research community on language inclusion. In this section, a benchmark means an ordered pair of BAs.

The first set of benchmarks consists of verification tasks defined together with the early versions of the RABIT tool [37]. The BAs are models of mutual exclusion algorithms [2], where in each benchmark one BA is the result of translating a set of guarded commands defining the protocol while the other BA translates a modified set of guarded commands, typically obtained by randomly weakening or strengthening one guard. The resulting BAs are on the binary alphabet $\{0,1\}$ and their sizes range from 20 to 7 963 states. Even though more details about transition labels and acceptance conditions are given [1, 2], it is unclear which basic properties this reduction satisfies, for instance, whether inclusion is preserved when the modified version of the protocol is the result of adding to the original version some "nop" statements. Moreover, we are not aware of any use of this reduction other than generating the RABIT examples.

Our second collection of benchmarks stems from an automated theorem prover for combinatorics on words called Pecan [34]. Here, BAs encode sets of solutions of predicates, hence logical implication between predicates reduces to a language inclusion problem between BAs. The benchmarks correspond to theorems of type $\forall x,\ P(x) \to Q(x)$ about Sturmian words [19]. We collected 58 benchmarks from Pecan for which inclusion holds, where these BAs have alphabets of varying size (from 3 to 256) and their sizes range from 1 to 21 395 states. The third collection of benchmarks stems from software verification. Ultimate Automizer (UA) [17, 18] is a well-known software model checker that verifies program correctness using automata-based reasoning, and that reduces termination problems to inclusion problems between BAs. Overall, we collected 600 benchmarks from UA for which inclusion holds. The BAs have alphabets of varying size (from 6 to 13 173) and sizes ranging from 3 to 6 972 states.

The addition of the Pecan and UA benchmarks significantly expands the set of available benchmarks while, at the same time, increases the diversity of their provenance. This set of benchmarks, which is available on GitHub [11], is biased towards instances where inclusion holds (as opposed to instances where inclusion does not hold). The rationale for this choice is that non-inclusion should be somehow viewed as a separate problem. This claim is supported by the existence of orthogonal approaches explicitly devoted to the non-inclusion problem [30] and specifically tailored approaches and optimizations within tools, like in RABIT. Nevertheless, let us remark that our approach decides the generic inclusion problem and has been evaluated on both positive and negative instances.

Table 1 Runtime in milliseconds on the BAs of Fig. 2. M/O means memory out.

Value of n	1	10	100	1 000	10 000	20 000	30 000	40 000	50 000
BAIT	34	48	100	531	92 102	342 526	821 234	1 284 618	2 074 829
RABIT	75	71	114	919	55 247	M/O	M/O	M/O	M/O

Table 2 Runtimes for RABIT benchmarks in millisec. GOAL$^-$ is Piterman inclusion algorithm without simulations (invocation flag `containment -m piterman`). M/O means memory out.

	Included									Not-included				
Tools	bk	bkv2	Fis	Fisv2	Fisv3	Fisv4	mcs	Pet	Φ	bkv3	Fisv5	Φv2	Φv3	Φv4
RABIT	2 220	4 552	2 260	213	3 985	1 271	49 193	71	136	2 697	6 002	334	265	287
ROLL	4 340	7 590	4 170	1 910	6 690	3 320	14 900	690	1 000	750	600	290	370	310
GOAL	88 320	71 090	128 680	3 500	41 620	18 120	456 480	1 380	2 260	2 450	12 580	1 480	1 510	2 260
GOAL$^-$	M/O	M/O	857 840	6 470	306 370	75 960	3 194 940	2 240	4 360	48 010	85 880	4 110	2 430	2 960
BAIT	1 180 770	M/O	M/O	1 153	654 385	M/O	M/O	321	278 794	6 347	M/O	1 749	997	65 629

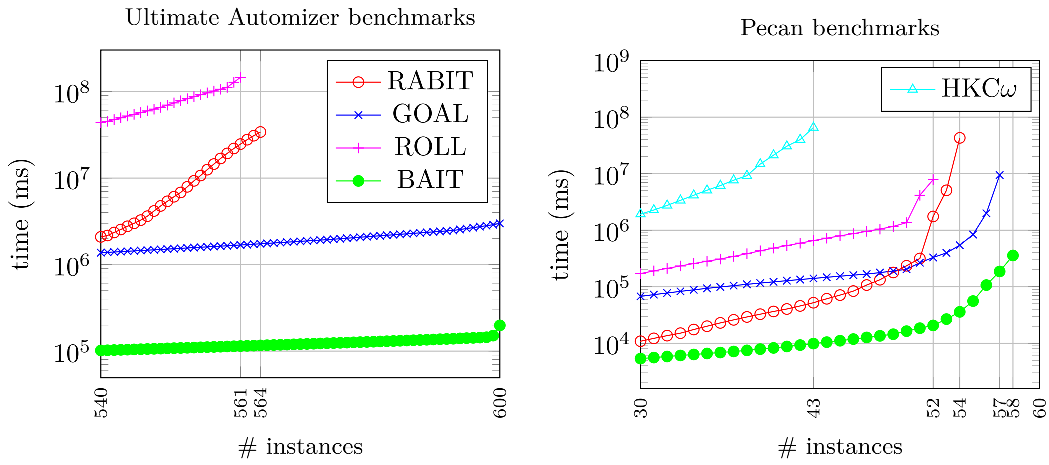
Tools. We implemented our state-based inclusion algorithm BAIncS in a tool called BAIT, developed in Java and which is available on GitHub [10]. We compared BAIT with the following language inclusion checking tools: RABIT 2.5.0, ROLL 1.0, GOAL (20200822), and HKCω (fall 2018). RABIT [7] implements a Ramsey-based algorithm and an advanced preprocessor using simulation relations. ROLL [27, 28] also uses the preprocessor of RABIT but then it relies on automata learning and word sampling techniques to decide inclusion. GOAL [41] implements a "complement-then-intersect-and-check-emptiness" approach using advanced complementation algorithms for BAs. HKCω [24] decides inclusion using up-to techniques. Further details on these tools are given in App. C.

Results. We ran our experiments on a server with 20 GB of RAM, 2 Xeon E5640 2.6 GHz CPUs and Debian stretch 64 bit. In what follows, "left"/"right" BAs refer, resp., to the automata on the left/right of a language inclusion instance.

We start with the following research question: *What is the impact in having separate qos for prefixes and periods?* To answer it, we first examine the performance of BAIT on the contrived family of examples of Fig. 2. In this set of instances, almost no computation is carried out in the fixpoints for the periods ($R_{\mathcal{A},\mathcal{B},p}$ of BAIncS), since they converge in one iteration. Tab. 1 displays the corresponding runtime comparison with RABIT, which processes prefixes and periods the same way. It turns out that for sufficiently large values of n, RABIT runs out of memory while BAIT safely terminates (in max 35 minutes).

Beyond the contrived family of BAs of Fig. 2, we claim that reasoning with separate qos for prefixes and periods gives an advantage to BAIT. Actually, we found that BAIT is the state-of-the-art on all but the RABIT benchmarks. On the RABIT benchmarks, Tab. 2 shows that BAIT runs out of memory on 4/9 of the included benchmarks and on 1/5 of the not-included benchmarks. On these benchmarks, simulation relations are key enablers for RABIT, ROLL and GOAL. Since the pair of BAs in each benchmark stems from two close revisions of the same mutual exclusion protocol, it turns out that the simulation relations being used retain enough information to dramatically lower the effort of showing inclusion (in many cases, these simulation relations alone are sufficient to show language inclusion).

To interpret these outcomes for BAIT, we looked at the graph structure of the "left" BAs of these RABIT benchmarks and we found that they roughly consist of one large strongly connected component (SCC): this is expected since these BAs model agents running a mutual exclusion protocol in an infinite loop. The computations of BAIT on these benchmarks are



(a) Plot not shown between 1 and 539 for clarity. HKCω not depicted: more than 60 memory out (8 GB virtual memory limit).

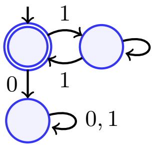
(b) Plot not shown between 1 and 29 for clarity.

Figure 3 Each benchmark has a timeout value of 12h. Survival plot with a logarithmic y axis and linear x axis. No plot for abscissa value x and tool r means that, for $60-x$ Pecan benchmarks (or $600-x$ for the case of Ultimate), r did not return an answer.

dominated by the fixpoints of $R_{\mathcal{A},\mathcal{B},p}$ for the periods, which compute over *sets of pairs of states*, as opposed to the fixpoints of $P_{\mathcal{A},\mathcal{B}}$ for the prefixes which compute over *sets of states*. Also, the computations for $R_{\mathcal{A},\mathcal{B},p}$ dominate the memory use. However, this scenario of BAs consisting of one large SCC does not occur for the other benchmarks: the "left" automata for Pecan and UA benchmarks tend to have few SCCs including final states and each of them are rather small. Here, BAIT is at an advantage because most computations are carried out on the fixpoints of $P_{\mathcal{A},\mathcal{B}}$ for the prefixes.

Because of the large number of available Pecan (60) and UA (600) benchmarks, we use survival plots for displaying our experimental results. Let us recall how to obtain them for a family of benchmarks $\{p_i\}_{i=1}^n$: (1) run the tool on each benchmark p_i and store its runtime t_i (or timeout event); (2) sort the t_i's in increasing order (discarding the timeouts); (3) plot the points $(t_1, 1), (t_1 + t_2, 2), \ldots$, and in general $(\sum_{i=1}^k t_i, k)$; (4) repeat for each tool under evaluation. The runtimes for BAIT include a phase of preprocessing that reduced the set of final states of the "left" BA while preserving the accepted language, in accordance with Remark 4.8. This preprocessing used the function `acc -min` of the tool GOAL, a polynomial time algorithm that relies on computing SCCs. The survival plots in compact form are depicted in Fig. 3 and with more detail in App. D.

These results show that BAIT is the state-of-the-art approach for the Pecan and UA benchmarks. They also show that GOAL performs quite well on the Pecan and UA benchmarks compared to RABIT and ROLL whose approaches are less efficacious. This is expected because both Pecan and UA rely on complementation for their decision procedure, so that they produce their "right" BAs through some heuristics to make them easy to complement (as confirmed to us by the developers of Pecan and UA). Indeed, we claim that GOAL's performance quickly degrades when the "right" BAs are hard to complement. Our claim is supported by Fig. 4 where GOAL and BAIT are compared on a contrived family of benchmarks based on Michel's family of hard to complement BAs (see [33] and [39, Theorem 5.3] for further details).



	$\Sigma = \{0,1,2,3\}$		$\Sigma = \{0,1,2,3,4\}$		$\Sigma = \{0,1,2,3,4,5\}$	
Value of n	BAIT	GOAL$^-$	BAIT	GOAL$^-$	BAIT	GOAL$^-$
3	45	13 550	$\not\subseteq$	$\not\subseteq$	$\not\subseteq$	$\not\subseteq$
4	45	14 040	77	9 187 680	$\not\subseteq$	$\not\subseteq$
5	46	13 120	89	9 148 710	225	T/O
10	54	13 840	100	T/O	291	T/O
100	123	17 020	282	T/O	1 301	T/O

Figure 4 Runtime in milliseconds using Michel's family for the "right" BAs (parameterized by n) and the depicted BA for the "left". GOAL$^-$ refers to `containment -m piterman` (as in Table 2). $\not\subseteq$ means not included, T/O is time out (12h).

7 Conclusion and Future Work

We designed a family of algorithms for the inclusion problem between ω-regular and ω-context-free languages into ω-regular languages, represented by automata. Our algorithms are conceptually simple: least fixpoint computations for the languages of finite prefixes and periods of ultimately periodic infinite words. The functions to iterate for these fixpoints are readily derived from the "left" automaton and the fixpoints converge in finitely many iterations thanks to a well-quasiorder abstraction on words. Finally, language inclusion is decided by a straightforward membership check. The height of the lattices of our least fixpoint computations allows us to derive some information about the worst case complexity of our algorithms. For each least fixpoint computation performed at line 3 of BAIncS, the worst case is adding exactly one element in a subset of $\wp(Q^2_{\mathcal{B}}) \times \wp(Q^2_{\mathcal{B}})$ to some entry of the $|Q_{\mathcal{A}}|$-dimensional vector at each iteration step, so that $|Q_{\mathcal{A}}| \times 2^{2|Q_B|^2}$ is an upper bound on N_2 in BAIncS. We leave as future work a detailed worst case complexity analysis of our algorithms. In practice, a simple Java implementation of our inclusion algorithm was competitive against state-of-the-art tools, thus showing the benefits of having separate well-quasiorders for prefixes and periods. We expect that this latter approach can be further refined using, for instance, family of right-congruences [31], paving the way to even more efficient inclusion algorithms.

References

1 Parosh Aziz Abdulla, Yu-Fang Chen, Lorenzo Clemente, Lukáš Holík, Chih-Duo Hong, Richard Mayr, and Tomáš Vojnar. Simulation Subsumption in Ramsey-Based Büchi Automata Universality and Inclusion Testing. In *Proc. Int. Conf. on Computer Aided Verification (CAV)*. Springer, 2010. `doi:10.1007/978-3-642-14295-6_14`.

2 Parosh Aziz Abdulla, Yu-Fang Chen, Lorenzo Clemente, Lukáš Holík, Chih-Duo Hong, Richard Mayr, and Tomáš Vojnar. Advanced Ramsey-Based Büchi Automata Inclusion Testing. In *Proc. Int. Conf. on Concurrency Theory (CONCUR)*. Springer LNCS, 2011. `doi:10.1007/978-3-642-23217-6_13`.

3 Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In *Proc. of the 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. ACM, 2013. `doi:10.1145/2429069.2429124`.

4 Filippo Bonchi and Damien Pous. Hacking nondeterminism with induction and coinduction. *Commun. ACM*, 58(2), 2015. `doi:10.1145/2713167`.

5 Hugues Calbrix, Maurice Nivat, and Andreas Podelski. Ultimately periodic words of rational ω-languages. In *Proc. Int. Symp. on Mathematical Foundations of Programming Semantics (MFPS)*, LNCS. Springer, 1994. `doi:10.1007/3-540-58027-1\_27`.

6 Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem. *Handbook of Model Checking*. Springer, 1st edition, 2018. `doi:10.1007/978-3-319-10575-8`.

7 Lorenzo Clemente and Richard Mayr. Efficient reduction of nondeterministic automata with application to language inclusion testing. *Logical Methods in Computer Science*, 15(1), 2019. doi:10.23638/LMCS-15(1:12)2019.

8 Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proc. 4th ACM Symp. on Principles of Programming Languages (POPL)*. ACM, 1977. doi:10.1145/512950.512973.

9 Patrick Cousot and Radhia Cousot. Systematic design of program analysis frameworks. In *Proc. of the 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL)*. ACM, 1979. doi:10.1145/567752.567778.

10 K. Doveri, P. Ganty, F. Parolini, and F. Ranzato. BAIT: Büchi Automata Inclusion Tester. https://github.com/parof/bait, 2021.

11 K. Doveri, P. Ganty, F. Parolini, and F. Ranzato. Büchi Automata benchmarks for language inclusion. https://github.com/parof/buchi-automata-benchmark, 2021.

12 Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proc. 14th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*, LNCS. Springer, 2016. doi:10.1007/978-3-319-46520-3_8.

13 Javier Esparza. Automata theory – An algorithmic approach. Lecture Notes, 2017. URL: https://www7.in.tum.de/~esparza/autoskript.pdf.

14 Seth Fogarty and Moshe Y. Vardi. Efficient Büchi Universality Checking. In *Proc. Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS. Springer, 2010. doi:10.1007/978-3-642-12002-2_17.

15 Pierre Ganty, Francesco Ranzato, and Pedro Valero. Language inclusion algorithms as complete abstract interpretations. In *Proc. 26th Int. Static Analysis Symposium (SAS)*, LNCS. Springer, 2019. doi:10.1007/978-3-030-32304-2_8.

16 Pierre Ganty, Francesco Ranzato, and Pedro Valero. Complete abstractions for checking language inclusion. *ACM Trans. on Computational Logic*, To appear, 2021. arXiv:1904.01388.

17 Matthias Heizmann, Yu-Fang Chen, Daniel Dietsch, Marius Greitschus, Jochen Hoenicke, Yong Li, Alexander Nutz, Betim Musa, Christian Schilling, Tanja Schindler, and Andreas Podelski. Ultimate Automizer and the search for perfect interpolants – (competition contribution). In *Proc. 24th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS. Springer, 2018. doi:10.1007/978-3-319-89963-3_30.

18 Matthias Heizmann, Jochen Hoenicke, and Andreas Podelski. Software model checking for people who love automata. In *Proc. Int. Conf. on Computer Aided Verification (CAV)*. Springer LNCS, 2013. doi:10.1007/978-3-642-39799-8_2.

19 Philipp Hieronymi, Dun Ma, Reed Oei, Luke Schaeffer, Zhengyao Lin, Christian Schulz, and Jeffrey Shallit. Decidability for Sturmian words, 2021. arXiv:2102.08207.

20 Martin Hofmann and Wei Chen. Abstract interpretation from Büchi automata. In *Proc. of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. ACM Press, 2014. doi:10.1145/2603088.2603127.

21 Takumi Kasai and Shigeki Iwata. Some problems in formal language theory known as decidable are proved EXPTIME complete, 1992. URL: https://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/0796-02.pdf.

22 Bakhadyr Khoussainov and Nerode, Anil. *Automata Theory and Its Applications*. Springer, 2001. doi:10.1007/978-1-4612-0171-7.

23 Denis Kuperberg, Laureline Pinault, and Damien Pous. HKCω: Coinductive algorithms for Büchi automata. http://perso.ens-lyon.fr/damien.pous/covece/hkcw/, 2018.

24 Denis Kuperberg, Laureline Pinault, and Damien Pous. Coinductive Algorithms for Büchi Automata. In *Proc. Int. Conf. on Developments in Language Theory (DLT)*, LNCS. Springer, 2019. doi:10.1007/978-3-030-24886-4_15.

25 Orna Kupferman. Automata Theory and Model Checking. In *Handbook of Model Checking*. Springer, 2018. doi:10.1007/978-3-319-10575-8_4.

26 Orna Kupferman and Moshe Y. Vardi. Verification of fair transition systems. In *Proc. Int. Conf. on Computer Aided Verification (CAV)*. Springer, 1996. doi:10.1007/3-540-61474-5_84.

27 Yong Li, Yu-Fang Chen, Lijun Zhang, and Depeng Liu. A novel learning algorithm for Büchi automata based on family of DFAs and classification trees. *Information and Computation*, 2020. doi:10.1016/j.ic.2020.104678.

28 Yong Li, Xuechao Sun, Andrea Turrini, Yu-Fang Chen, and Junnan Xu. ROLL 1.0: ω-regular language learning library. In *Proc. 25th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS. Springer, 2019. doi:10.1007/978-3-030-17462-0_23.

29 Yong Li and Andrea Turini. Roll library: Regular Omega Language Learning library. https://github.com/ISCAS-PMC/roll-library, 2020.

30 Yong Li, Andrea Turrini, Xuechao Sun, and Lijun Zhang. Proving non-inclusion of Büchi automata based on Monte Carlo sampling. In *Proc. 14th Int. Symp. on Automated Technology for Verification and Analysis (ATVA)*. Springer, 2020. doi:10.1007/978-3-030-59152-6_26.

31 Oded Maler and Ludwig Staiger. On syntactic congruences for ω-languages. Technical report, Verimag, France, 2008. URL: http://www-verimag.imag.fr/~maler/Papers/congr.pdf.

32 Roland Meyer, Sebastian Muskalla, and Elisabeth Neumann. Liveness verification and synthesis: New algorithms for recursive programs, 2017. arXiv:1701.02947.

33 M. Michel. Complementation is more difficult with automata on infinite words. Technical report, CNET, Paris, 1988.

34 Reed Oei, Dun Ma, Christian Schulz, and Philipp Hieronymi. Pecan: An automated theorem prover for automatic sequences using Büchi automata, 2021. arXiv:2102.01727.

35 Dominique Perrin and Jean Eric Pin. *Infinite Words: Automata, Semigroups, Logic and Games*. Number 141 in Pure and Applied Mathematics Series. Elsevier, Amsterdam ; Boston, 1st edition, 2004.

36 Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Logical Methods in Computer Science*, 3(3), 2007. doi:10.2168/lmcs-3(3:5)2007.

37 RABIT/Reduce: Tools for language inclusion testing and reduction of nondeterministic Büchi automata and NFA. http://www.languageinclusion.org/doku.php?id=tools. Accessed: 2021-01-29.

38 Roman R. Redziejowski. An improved construction of deterministic omega-automaton using derivatives. *Fundamenta Informaticae*, 119(3–4), 2012. doi:10.3233/FI-2012-744.

39 Wolfgang Thomas. Languages, Automata, and Logic. In *Handbook of Formal Languages: Volume 3 Beyond Words*. Springer, 1997. doi:10.1007/978-3-642-59126-6_7.

40 Ming-Hsien Tsai, Seth Fogarty, Moshe Vardi, and Yih-Kuen Tsay. State of Büchi complementation. *Logical Methods in Computer Science*, 10(4), 2014. doi:10.2168/lmcs-10(4:13)2014.

41 Ming-Hsien Tsai, Yih-Kuen Tsay, and Yu-Shiang Hwang. GOAL for Games, Omega-Automata, and Logics. In *Proc. Int. Conf. on Computer Aided Verification (CAV)*. Springer, 2013. doi:10.1007/978-3-642-39799-8_62.

42 William M. Waite and Gerhard Goos. *Compiler Construction*. Springer-Verlag, New York, USA, 1984.

A Generalised Word-Based Algorithm

We give a generalised word-based algorithm gBAIncW, briefly explained in Section 4.1, which computes sequences of pruned Kleene iterates $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_{p,n}\}_{n\in\mathbb{N}}$, for each $p \in F$. We obtain the correcteness of BAIncW as a consequence of the correcteness of gBAIncW.

▶ **Theorem A.1.** *Given all the required input data,* gBAIncW *decides* $L^\omega(\mathcal{A}) \subseteq M$*.*

gBAIncW Word-based algorithm for checking $L^\omega(\mathcal{A}) \subseteq M$.

```
Data: Büchi automaton $\mathcal{A} = (Q, \delta, i_\mathcal{A}, F)$
Data: Procedure deciding $uv^\omega \in^? M$ given $u, v \in \Sigma^*$
Data: Decidable right-monotonic wqos $\le, \preccurlyeq$ s.t. $\rho_{\le \times \preccurlyeq}(I_M) = I_M$
Data: For each $p \in F$ and $n \in \mathbb{N}$, sequences $\{X_n\}_{n\in\mathbb{N}}$ and $\{Y_{p,n}\}_{n\in\mathbb{N}}$ in $\wp(\Sigma^*)^{|Q|}$ s.t.
      $\rho_\le(P^n_\mathcal{A}(\emptyset)) = \rho_\le(X_n)$ and $\rho_\preccurlyeq(R^n_{\mathcal{A},p}(\emptyset)) = \rho_\preccurlyeq(Y_{p,n})$.
Compute $X_{N_1}$ with least $N_1$ s.t. $\forall q \in Q$, $\rho_\le((X_{N_1+1})_q) \subseteq \rho_\le((X_{N_1})_q)$
foreach $p \in F$ do
    Compute $Y_{p,N_2}$ with least $N_2$ s.t. $\forall q \in Q$, $\rho_\preccurlyeq((Y_{p,N_2+1})_q) \subseteq \rho_\le((Y_{p,N_2})_q)$
    foreach $u \in (X_{N_1})_p$, $v \in (Y_{p,N_2})_p$ do
        if $uv^\omega \notin M$ then return false;
return true;
```

B State-Based Algorithm for ω-context-free $\subseteq$ ω-regular

We derive a state-based inclusion algorithm that, given a BPDA $\mathcal{P} = (Q_\mathcal{P}, \Gamma, \delta_\mathcal{P}, i_\mathcal{P}, F_\mathcal{P})$ and a BA $\mathcal{B} = (Q_\mathcal{B}, \delta_\mathcal{B}, i_\mathcal{B}, F_\mathcal{B})$, decides whether $L^\omega(\mathcal{P}) \subseteq L^\omega(\mathcal{B})$ holds or not by operating on the states of $\mathcal{P}$ and $\mathcal{B}$ only. Similarly to the ω-regular case, words and operations/tests on words are abstracted, resp., into states and operations/tests on states, using the state-based qos derived from $\mathcal{B}$, as explained in Section 3. Recall that in the context-free case we need qos that are both right- and left- monotonic. Hence, we consider the pair of qos $\trianglelefteq^\mathcal{B}, \preccurlyeq^\mathcal{B}$ (see Section 3).

Given a CFG $\mathcal{G} = (V, P)$ in CNF, we define the functions $R_{1,\mathcal{G}}$ over $\wp(Q^2_\mathcal{B})^V$ and $R_{2,\mathcal{G}}$ over $(\wp(\wp(Q^2_\mathcal{B}) \times \wp(Q^2_\mathcal{B})))^V$ as follows:

$$
\begin{aligned}
R^1_\mathcal{G}(S) &\triangleq \langle \{x \circ y \mid \exists X_j \to X_k X_l \in P, x \in S_k \wedge y \in S_l\} \rangle_{j\in[0,n]} \;, \\
R^2_\mathcal{G}(S) &\triangleq \langle \{(x_1 \circ y_1, (x_1 \circ y_2) \cup (x_2 \circ y_1)) \mid \exists X_j \to X_k X_l \in P, \\
&\qquad\qquad (x_1, x_2) \in S_k, (y_1, y_2) \in S_l\} \rangle_{j\in[0,n]} \;.
\end{aligned}
$$

Let us define the vectors $b^1_\mathcal{G} \in \wp(Q^2_\mathcal{B})^V$ and $b^2_\mathcal{G} \in (\wp(\wp(Q^2_\mathcal{B}) \times \wp(Q^2_\mathcal{B})))^V$ as follows:

$$
\begin{aligned}
b^1_\mathcal{G} &\triangleq \langle \{c^\mathcal{B}[\beta] \mid X_j \to \beta, \beta \in \Sigma \cup \{\epsilon\}\} \rangle_{j\in[0,n]} \;, \\
b^2_\mathcal{G} &\triangleq \langle \{(c^\mathcal{B}[\beta], f^\mathcal{B}[\beta]) \mid X_j \to \beta, \beta \in \Sigma \cup \{\epsilon\}\} \rangle_{j\in[0,n]} \;.
\end{aligned}
$$

Let $\mathcal{P}^1_{q\gamma}$ and $\mathcal{P}^2_{q\gamma}$, for each $q \in Q_\mathcal{P}$ and $\gamma \in \Gamma$, be the two PDAs defined from $\mathcal{P}$ and such that the ultimately periodic words generated by the pairs in $\bigcup_{(q,\gamma)\in Q\times\Gamma} L^*(\mathcal{P}^1_{q,\gamma}) \times L^*(\mathcal{P}^2_{q,\gamma})$ coincide with the ultimately periodic words in $L^\omega(\mathcal{P})$. Let $\mathcal{G}^1_{q\gamma} \triangleq \text{PDA2CFG}(\mathcal{P}^1_{q\gamma})$ and $\mathcal{G}^2_{q\gamma} \triangleq \text{PDA2CFG}(\mathcal{P}^2_{q\gamma})$, where PDA2CFG is a procedure to convert a PDA into a CFG in CNF. For each $q \in Q_\mathcal{P}$ and each $\gamma \in \Gamma$, we define the functions $P_{q\gamma\mathcal{B}} \triangleq \lambda X. b^1_{\mathcal{G}^1_{q\gamma}} \cup R^1_{\mathcal{G}^1_{q\gamma}}(X)$ and $R_{q\gamma\mathcal{B}} \triangleq \lambda X. b^2_\mathcal{G} \cup R^2_{\mathcal{G}^2_{q\gamma}}(X)$. Let us define the following state-abstraction of the membership test:

$$
\text{Inc}^{\mathcal{B}cf}(x, y_1, y_2) \triangleq \exists p, q \in Q_\mathcal{B}, (i_\mathcal{B}, p) \in x \wedge (p, q) \in y_1^* \wedge (q, q) \in y_1^* \circ y_2 \circ y_1^* \;.
$$

▶ **Lemma B.1.** $uv^\omega \in L^\omega(\mathcal{B}) \iff \text{Inc}^{\mathcal{B}cf}(c^\mathcal{B}[u], c^\mathcal{B}[v], f^\mathcal{B}[v])$.

Algorithm BPDAIncS: State-based algorithm for $L^\omega(\mathcal{P}) \subseteq L^\omega(\mathcal{B})$.

```
Data: BPDA $\mathcal{P} = (Q, \Gamma, \delta, q_0, Z_0, F)$ and BA $\mathcal{B} = (Q_\mathcal{B}, \delta_\mathcal{B}, i_\mathcal{B}, F_\mathcal{B})$
foreach $q \in Q$, $\gamma \in \Gamma$ do
    $\mathcal{G}_1 := \mathrm{PDA2CFG}(\mathcal{P}^1_{[q\gamma]})$; $\mathcal{G}_2 := \mathrm{PDA2CFG}(\mathcal{P}^2_{[q\gamma]})$;
    Compute $\lfloor P^{N_1}_{q\gamma\mathcal{B}} \rfloor$ with least $N_1$ s.t. $\forall j \in V_{\mathcal{G}^1_{q\gamma}}$, $\rho_\subseteq((P^{N_1+1}_{q\gamma\mathcal{B}}(\emptyset))_j) \subseteq \rho_\subseteq((P^{N_1}_{q\gamma\mathcal{B}}(\emptyset))_j)$;
    Compute $\lfloor R^{N_2}_{q\gamma\mathcal{B}} \rfloor$ with least $N_2$ s.t. $\forall j \in V_{\mathcal{G}^2_{q\gamma}}$, $\rho_{\subseteq^2}((R^{N_2+1}_{q\gamma\mathcal{B}}(\emptyset))_j) \subseteq \rho_{\subseteq^2}((R^{N_2}_{q\gamma\mathcal{B}}(\emptyset))_j)$;
    foreach $x \in (\lfloor P^{N_1}_{q\gamma\mathcal{B}} \rfloor)_0$, $(y_1, y_2) \in (\lfloor R^{N_2}_{q\gamma\mathcal{B}} \rfloor)_0$ do
        if $\neg \mathrm{Inc}^{\mathcal{B}cf}(x, y_1, y_2)$ then return false;
return true;
```

▶ **Theorem B.2.** *Given a BPDA $\mathcal{P}$ and BA $\mathcal{B}$,* BPDAIncS *decides $L^\omega(\mathcal{P}) \subseteq L^\omega(\mathcal{B})$.*

C Language Inclusion Checking Tools

RABIT [7] consists of about 20K lines of Java code and its source code is publicly available [37]. To check a language inclusion RABIT combines several techniques controlled via command line options. In our experiments we ran RABIT with options `-fast -jf` which RABIT states as providing the "best performance". Roughly speaking, RABIT performs the following operations: (1) Removing dead states and minimizing the automata with simulation-based techniques, thus yielding a smaller instance; (2) Witnessing inclusion by simulation already during the minimization phase; (3) Using the Ramsey-based method to witness inclusion or non-inclusion.

ROLL [27, 28] contains an inclusion checker that does a preprocessing similar to that of RABIT and then relies on automata learning and word sampling techniques to decide inclusion. ROLL consists of about 19K lines of Java code which is publicly available [29].

GOAL [41] contains several language inclusion checkers available with multiple options. We used the Piterman check (`containment -m piterman -sim -pre` on the command line) that constructs on-the-fly the intersection of the "left" BA and the complement of the "right" BA which is itself built on-the-fly by the Piterman construction [36]. The options `-sim -pre` compute and use simulation relations to further improve performance. The Piterman check was deemed the "best effort" (cf. [7, Section 9.1] and [40]) among the inclusion checkers provided in GOAL. GOAL is written in Java and the source code of the release we used is not publicly available.

HKCω [24] includes an inclusion checker using the so-called up-to techniques. HKCω consists of 3K lines of OCaml code which is publicly available [23]. Up-to techniques form the state-of-the-art approach to decide equivalence for languages of finite words given by finite state automata [3, 4]. The extension of up-to techniques to ω-words has been implemented in HKCω, although only partially. Indeed, as stated in the code documentation, even if up-to techniques have been defined for both prefixes and periods of ultimately periodic words, HKCω only implements them for prefixes. HKCω also includes some preprocessing of the BAs using simulation relations.

As far as we know all these implementations are sequential except for RABIT which, using the `-jf` option, performs some computations in a separate thread.

BAIT is our implementation of the BAIncS algorithm defined in Section 5. BAIT consists of less than 1 750 lines of Java code. BAIT relies exclusively on a few standard packages from the Java SE Platform, notably standard collections such as HashSet or HashMap. One of the design goals of BAIT was to have simple and unencumbered code. Unlike RABIT, HKCω, ROLL and GOAL, BAIT does not compute or exploit simulation relations. Also, BAIT is implemented as a purely sequential algorithm although some computations are easily parallelizable such as the fixpoints for the prefixes and for the periods.

SPOT We did not consider the Spot tool [12] in our evaluation because we believe GOAL is a better fit in our setting as we argue below. First, Spot works with a symbolic alphabet where symbols are encoded using Boolean propositions, and sets of symbols are represented and processed using OBDDs. We used GOAL in the classical alphabet mode where symbols are explicitly represented as in ROLL, RABIT and BAIT. Second, the inclusion algorithm of Spot complements the "right" BA using Redziejowski's method with some additional optimizations including simulation-based optimizations [12]. GOAL implements Piterman's complementation method [36], which inspired that of Redziejowski [38]. The Piterman's method of GOAL also offers simulation-based optimizations and, furthermore, GOAL specialized Piterman's method to the inclusion problem by constructing on-the-fly the intersection of the "left" automaton and the complement of the "right" automaton constructed on-the-fly by the Piterman's method [40]. Finally, Spot is written in C++ while GOAL is written in Java as ROLL, RABIT and BAIT, thus making their runtime comparison more meaningful.

Experimental Setup. We ran our experiments on a server with 20 GB of RAM, 2 Xeon E5640 2.6 GHz CPUs and Debian stretch 64 bit. We used openJDK 11.0.9.1 2020-11-04 when compiling Java code and ran the JVM with default options. For RABIT and BAIT the execution time is computed using timers internal to their implementations. For ROLL and GOAL the execution time is given by the "real" value of the `time(1)` command.

D Detailed Graphs of the Experimental Comparison

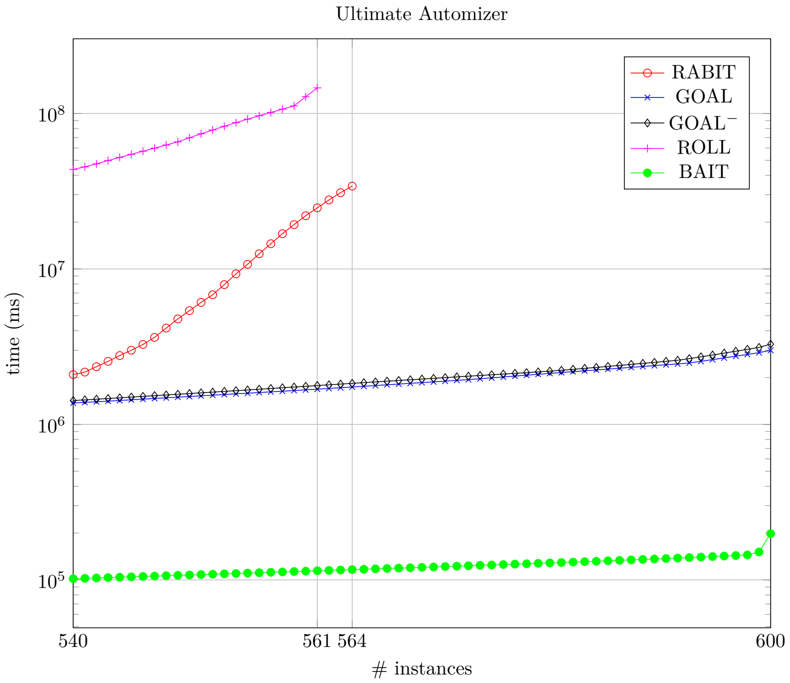


Figure 5 Survival plot with a logarithmic y axis and linear x axis. Plot not depicted between 1 and 539 for clarity. Each benchmark has a timeout value of 12h. No plot for abscissa value x and tool r means that, for $600-x$ benchmarks, r did not return an answer (i.e. it either ran out of memory or time). HKCω not depicted: more than 60 memory out (8 GB virtual memory limit).

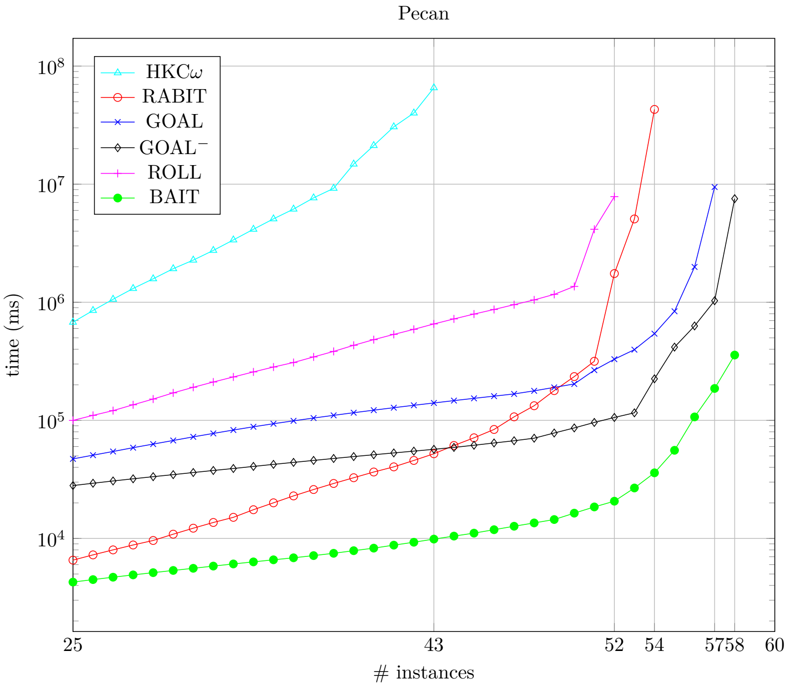


Figure 6 Survival plot with a logarithmic y axis and linear x axis. Plot not depicted between 1 and 24 for clarity. Each benchmark has a timeout value of 12h. No plot for abscissa value x and tool r means that, for $60-x$ Pecan benchmarks, r did not return an answer (i.e., it either ran out of memory or time).

Nominal Büchi Automata with Name Allocation

Henning Urbat
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Daniel Hausmann
Gothenburg University, Göteborg, Sweden

Stefan Milius
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Lutz Schröder
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Abstract

Infinite words over infinite alphabets serve as models of the temporal development of the allocation and (re-)use of resources over linear time. We approach ω-languages over infinite alphabets in the setting of nominal sets, and study languages of *infinite bar strings*, i.e. infinite sequences of names that feature binding of fresh names; binding corresponds roughly to reading letters from input words in automata models with registers. We introduce *regular nominal nondeterministic Büchi automata* (*Büchi RNNAs*), an automata model for languages of infinite bar strings, repurposing the previously introduced *RNNAs* over finite bar strings. Our machines feature explicit binding (i.e. resource-allocating) transitions and process their input via a Büchi-type acceptance condition. They emerge from the abstract perspective on name binding given by the theory of nominal sets. As our main result we prove that, in contrast to most other nondeterministic automata models over infinite alphabets, language inclusion of Büchi RNNAs is decidable and in fact elementary. This makes Büchi RNNAs a suitable tool for applications in model checking.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects

Keywords and phrases Data languages, infinite words, nominal sets, inclusion checking

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.4

Related Version *Full Version*: http://arxiv.org/abs/2107.03213

Funding *Henning Urbat*: Supported by Deutsche Forschungsgemeinschaft (DFG) under project SCHR 1118/15-1.
Daniel Hausmann: Supported by Deutsche Forschungsgemeinschaft (DFG) under project MI 717/7-1 and by the European Research Council (ERC) under project 772459.
Stefan Milius: Supported by Deutsche Forschungsgemeinschaft (DFG) under project MI 717/7-1.
Lutz Schröder: Supported by Deutsche Forschungsgemeinschaft (DFG) under project SCHR 1118/15-1.

1 Introduction

Classical automata models and formal languages for finite words over finite alphabets have been extended to infinity in both directions: Infinite words model the long-term temporal development of systems, while infinite alphabets model *data*, such as nonces [19], object identities [14], or abstract resources [5]. We approach *data ω-languages*, i.e. languages of infinite words over infinite alphabets, in the setting of nominal sets [25] where elements of sets are thought of as carrying (finitely many) *names* from a fixed countably infinite reservoir. Following the paradigm of nominal automata theory [3], we take the set of names as the alphabet; we work with infinite words containing explicit name binding in the spirit of previous work on nominal formalisms for finite words such as nominal Kleene algebra [11] and regular nondeterministic nominal automata (RNNAs) [26]. Name binding may be viewed

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 4; pp. 4:1–4:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

as the allocation of resources, or (yet) more abstractly as an operation that reads fresh names from the input. We refer to such infinite words as *infinite bar strings*, in honour of the vertical bar notation we employ for name binding. In the present paper, we introduce a notion of nondeterministic nominal automata over infinite bar strings, and show that it admits inclusion checking in elementary complexity.

Specifically, we reinterpret the mentioned RNNAs to accept infinite rather than finite bar strings by equipping them with a Büchi acceptance condition; like over finite alphabets, automata with more expressive acceptance conditions including Muller acceptance can be translated into the basic (nondeterministic) Büchi model. Our mentioned main result then states that language inclusion of Büchi RNNAs is decidable in parametrized polynomial space [27], with a parameter that may be thought as the number of registers. This is in sharp contrast to other nondeterministic automata models for infinite words over infinite alphabets, which sometimes have decidable emptiness problems but whose inclusion problems are typically either undecidable or of prohibitively high complexity even under heavy restrictions (inclusion is not normally reducible to emptiness since nondeterministic models typically do not determinize, and in fact tend to fail to be closed under complement); details are in the related work section.

Infinite bar strings can be concretized to infinite strings of names (i.e. essentially to data words) by interpreting name binding as reading either globally fresh letters (in which case Büchi RNNAs may essentially be seen as a variant of session automata [4] for infinite words) or locally fresh letters. Both interpretations arise from disciplines of α-renaming as known from λ-calculus [1], with global freshness corresponding to a discipline of *clean* naming where bound names are never shadowed, and local freshness corresponding to an unrestricted naming discipline that does allow shadowing. The latter implies that local freshness can only be enforced w.r.t. names that are expected to be seen again later in the word. It is precisely this fairly reasonable-sounding restriction that buys the comparatively low computational complexity of the model, which on the other hand allows full nondeterminism (and, e.g., accepts the language 'some letter occurs infinitely often', which is not acceptable by deterministic register-based models) and unboundedly many registers. Büchi RNNAs thus provide a reasonably expressive automata model for infinite data words whose main reasoning problems are decidable in elementary complexity.

Related Work. Büchi RNNAs generally adhere to the paradigm of register automata, which in their original incarnation over finite words [16] are equivalent to the nominal automaton model of nondeterministic orbit-finite automata [3]. Ciancia and Sammartino [5] study *deterministic* nominal Muller automata accepting infinite strings of names, and show Boolean closure and decidability of inclusion. This model is incomparable to ours; details are in Section 7. For alternating register automata over infinite data words, emptiness is undecidable even when only one register is allowed (which over finite data words does ensure decidability) [9]. For the one-register safety fragment of the closely related logic Freeze LTL, inclusion (i.e. refinement) is decidable, but even the special case of validity is not primitive recursive [21], in particular not elementary.

Many automata models and logics for data words deviate rather substantially from the register paradigm, especially models in the vicinity of data automata [2], whose emptiness problem is decidable but at least as hard as Petri net reachability, which by recent results is not elementary [7], in fact Ackermann-complete [8, 22]. For weak Büchi data automata [17], the emptiness problem is decidable in elementary complexity. Similarly, Büchi generalized data automata [6] have a decidable emptiness problem; their Büchi component is deterministic.

(Throughout, nothing appears to be said about inclusion of data automata.) Variable finite automata [15] apply to both finite and infinite words; in both versions, the inclusion problem of the nondeterministic variant is undecidable.

2 Preliminaries: Nominal Sets

Nominal sets form a convenient formalism for dealing with names and freshness; for our present purposes, names play the role of data. We briefly recall basic notions and facts and refer to [25] for a comprehensive introduction. Fix a countably infinite set $\mathbb{A}$ of *names*, and let $\mathsf{Perm}(\mathbb{A})$ denote the group of finite permutations on $\mathbb{A}$, which is generated by the *transpositions* $(a\,b)$ for $a \neq b \in \mathbb{A}$ (recall that $(a\,b)$ just swaps a and b). A *nominal set* is a set X equipped with a (left) group action $\mathsf{Perm}(\mathbb{A}) \times X \to X$, denoted $(\pi, x) \mapsto \pi \cdot x$, such that every element $x \in X$ has a finite *support* $S \subseteq \mathbb{A}$, i.e. $\pi \cdot x = x$ for every $\pi \in \mathsf{Perm}(\mathbb{A})$ such that $\pi(a) = a$ for all $a \in S$. Every element x of a nominal set X has a least finite support, denoted $\mathsf{supp}(x)$. Intuitively, one should think of X as a set of syntactic objects (e.g. strings, λ-terms, programs), and of $\mathsf{supp}(x)$ as the set of names needed to describe an element $x \in X$. A name $a \in \mathbb{A}$ is *fresh* for x, denoted $a \# x$, if $a \notin \mathsf{supp}(x)$. The *orbit* of an element $x \in X$ is given by $\{\pi \cdot x : \pi \in \mathsf{Perm}(\mathbb{A})\}$. The orbits form a partition of X. The nominal set X is *orbit-finite* if it has only finitely many orbits.

Putting $\pi \cdot a = \pi(a)$ makes $\mathbb{A}$ into a nominal set. Moreover, $\mathsf{Perm}(\mathbb{A})$ acts on subsets $A \subseteq X$ of a nominal set X by $\pi \cdot A = \{\pi \cdot x : x \in A\}$. A subset $A \subseteq X$ is *equivariant* if $\pi \cdot A = A$ for all $\pi \in \mathsf{Perm}(\mathbb{A})$. More generally, it is *finitely supported* if it has finite support w.r.t. this action, i.e. there exists a finite set $S \subseteq \mathbb{A}$ such that $\pi \cdot A = A$ for all $\pi \in \mathsf{Perm}(\mathbb{A})$ such that $\pi(a) = a$ for all $a \in S$. The set A is *uniformly finitely supported* if $\bigcup_{x \in A} \mathsf{supp}(x)$ is a finite set. This implies that A is finitely supported, with least support $\mathsf{supp}(A) = \bigcup_{x \in A} \mathsf{supp}(x)$ [10, Theorem 2.29]. (The converse does not hold, e.g. the set $\mathbb{A}$ is finitely supported but not uniformly finitely supported.) Uniformly finitely supported orbit-finite sets are always finite (since an orbit-finite set contains only finitely many elements with a given finite support). We respectively denote by $\mathcal{P}_{\mathsf{ufs}}(X)$ and $\mathcal{P}_{\mathsf{fs}}(X)$ the nominal sets of (uniformly) finitely supported subsets of a nominal set X.

A map $f\colon X \to Y$ between nominal sets is *equivariant* if $f(\pi \cdot x) = \pi \cdot f(x)$ for all $x \in X$ and $\pi \in \mathsf{Perm}(\mathbb{A})$. Equivariance implies $\mathsf{supp}(f(x)) \subseteq \mathsf{supp}(x)$ for all $x \in X$. The function supp itself is equivariant, i.e. $\mathsf{supp}(\pi \cdot x) = \pi \cdot \mathsf{supp}(x)$ for $\pi \in \mathsf{Perm}(\mathbb{A})$. Hence $|\,\mathsf{supp}(x_1)| = |\,\mathsf{supp}(x_2)|$ whenever x_1, x_2 are in the same orbit of a nominal set.

We denote by **Nom** the category of nominal sets and equivariant maps. The object maps $X \mapsto \mathcal{P}_{\mathsf{ufs}}(X)$ and $X \mapsto \mathcal{P}_{\mathsf{fs}}(X)$ extend to endofunctors $\mathcal{P}_{\mathsf{ufs}}\colon \mathbf{Nom} \to \mathbf{Nom}$ and $\mathcal{P}_{\mathsf{fs}}\colon \mathbf{Nom} \to \mathbf{Nom}$ sending an equivariant map $f\colon X \to Y$ to the map $A \mapsto f[A]$.

The coproduct $X + Y$ of nominal sets X and Y is given by their disjoint union with the group action inherited from the two summands. Similarly, the product $X \times Y$ is given by the cartesian product with the componentwise group action; we have $\mathsf{supp}(x, y) = \mathsf{supp}(x) \cup \mathsf{supp}(y)$. Given a nominal set X equipped with an equivariant equivalence relation, i.e. an equivalence relation $\sim$ that is equivariant as a subset $\sim \subseteq X \times X$, the quotient $X/{\sim}$ is a nominal set under the expected group action defined by $\pi \cdot [x]_\sim = [\pi \cdot x]_\sim$.

A key role in the technical development is played by *abstraction sets*, which provide a semantics for binding mechanisms [12]. Given a nominal set X, an equivariant equivalence relation $\sim$ on $\mathbb{A} \times X$ is defined by $(a, x) \sim (b, y)$ iff $(a\,c) \cdot x = (b\,c) \cdot y$ for some (equivalently, all) fresh c. The *abstraction set* $[\mathbb{A}]X$ is the quotient set $(\mathbb{A} \times X)/{\sim}$. The $\sim$-equivalence class of $(a, x) \in \mathbb{A} \times X$ is denoted by $\langle a \rangle x \in [\mathbb{A}]X$. We may think of $\sim$ as an abstract notion of α-equivalence, and of $\langle a \rangle$ as binding the name a. Indeed we have $\mathsf{supp}(\langle a \rangle x) = \mathsf{supp}(x) \setminus \{a\}$ (while $\mathsf{supp}(a, x) = \{a\} \cup \mathsf{supp}(x)$), as expected in binding constructs.

The object map $X \mapsto [\mathbb{A}]X$ extends to an endofunctor $[\mathbb{A}]\colon \mathbf{Nom} \to \mathbf{Nom}$ sending an equivariant map $f\colon X \to Y$ to the equivariant map $[\mathbb{A}]f\colon [\mathbb{A}]X \to [\mathbb{A}]Y$ given by $\langle a \rangle x \mapsto \langle a \rangle f(x)$ for $a \in \mathbb{A}$ and $x \in X$.

3 The Notion of α-Equivalence for Bar Strings

In the following we investigate automata consuming input words over the infinite alphabet

$$\overline{\mathbb{A}} := \mathbb{A} \cup \{|a : a \in \mathbb{A}\}.$$

A *finite bar string* is a finite word $\sigma_1\sigma_2\cdots\sigma_n$ over $\overline{\mathbb{A}}$, and *infinite bar string* is an infinite word $\sigma_1\sigma_2\sigma_3\cdots$ over $\overline{\mathbb{A}}$. We denote the sets of finite and infinite bar strings by $\overline{\mathbb{A}}^*$ and $\overline{\mathbb{A}}^\omega$, respectively. Given $w \in \overline{\mathbb{A}}^* \cup \overline{\mathbb{A}}^\omega$ the set of *names* in w is defined by

$$\mathsf{N}(w) = \{a \in \mathbb{A} : \text{the letter } a \text{ or } |a \text{ occurs in } w\}.$$

An infinite bar string w is *finitely supported* if $\mathsf{N}(w)$ is a finite set; we let $\overline{\mathbb{A}}^\omega_{\mathsf{fs}} \subseteq \overline{\mathbb{A}}^\omega$ denote the set of finitely supported infinite bar strings. Note that $\overline{\mathbb{A}}^*$ and $\overline{\mathbb{A}}^\omega_{\mathsf{fs}}$ are nominal sets w.r.t. the group action defined pointwise. The least support of a bar string is its set of names.

A bar string containing only letters from $\mathbb{A}$ is called a *data word*. We denote by $\mathbb{A}^* \subseteq \overline{\mathbb{A}}^*$, $\mathbb{A}^\omega \subseteq \overline{\mathbb{A}}^\omega$ and $\mathbb{A}^\omega_{\mathsf{fs}} \subseteq \overline{\mathbb{A}}^\omega_{\mathsf{fs}}$ the sets of finite data words, infinite data words, and finitely supported infinite data words, respectively.

We interpret $|a$ as binding the name a to the right. Accordingly, a name $a \in \mathbb{A}$ is said to be *free* in a bar string $w \in \overline{\mathbb{A}}^* \cup \overline{\mathbb{A}}^\omega$ if (i) the letter a occurs in w, and (ii) the first occurrence of a is not preceded by any occurrence of $|a$. For instance, the name a is free in $a|aba$ but not free in $|aaba$, while the name b is free in both bar strings. We put

$$\mathsf{FN}(w) = \{a \in \mathbb{A} : a \text{ is free in } w\}.$$

We obtain a natural notion of α-equivalence for both finite and infinite bar strings; the finite case is taken from [26].

▶ **Definition 3.1** (α-equivalence)**.** Let $=_\alpha$ be the least equivalence relation on $\overline{\mathbb{A}}^*$ such that

$$x|av =_\alpha x|bw \qquad \text{for all } a, b \in \mathbb{A} \text{ and } x, v, w \in \overline{\mathbb{A}}^* \text{ such that } \langle a \rangle v = \langle b \rangle w.$$

This extends to an equivalence relation $=_\alpha$ on $\overline{\mathbb{A}}^\omega$ given by

$$v =_\alpha w \qquad \text{iff} \qquad v_n =_\alpha w_n \quad \text{for all } n \in \mathbb{N},$$

where v_n and w_n are the prefixes of length n of v and w. We denote by $\overline{\mathbb{A}}^*/{=_\alpha}$ and $\overline{\mathbb{A}}^\omega/{=_\alpha}$ the sets of α-equivalence classes of finite and infinite bar strings, respectively, and we write $[w]_\alpha$ for the α-equivalence class of $w \in \overline{\mathbb{A}}^* \cup \overline{\mathbb{A}}^\omega$.

▶ **Remark 3.2.**

1. For any $v, w \in \overline{\mathbb{A}}^*$ the condition $\langle a \rangle v = \langle b \rangle w$ holds if and only if

$$a = b \text{ and } v = w, \qquad \text{or} \qquad b \# v \text{ and } (a\, b) \cdot v = w.$$

2. The α-equivalence relation is a *left congruence*:

$$v =_\alpha w \quad \text{implies} \quad xv =_\alpha xw \qquad\qquad \text{for all } v, w \in \overline{\mathbb{A}}^* \cup \overline{\mathbb{A}}^\omega \text{ and } x \in \overline{\mathbb{A}}^*.$$

   Moreover, the *right cancellation property* holds:

$$vx =_\alpha wx \quad \text{implies} \quad v =_\alpha w \qquad\qquad \text{for all } v, w \in \overline{\mathbb{A}}^* \text{ and } x \in \overline{\mathbb{A}}^* \cup \overline{\mathbb{A}}^\omega.$$

3. The equivalence relation $=_\alpha$ is equivariant. Therefore, both $\overline{\mathbb{A}}^*/{=_\alpha}$ and $\overline{\mathbb{A}}_{\mathsf{fs}}^\omega/{=_\alpha}$ are nominal sets with the group action $\pi \cdot [w]_\alpha = [\pi \cdot w]_\alpha$ for $\pi \in \mathsf{Perm}(\mathbb{A})$ and $w \in \overline{\mathbb{A}}^* \cup \overline{\mathbb{A}}_{\mathsf{fs}}^\omega$. The least support of $[w]_\alpha$ is the set $\mathsf{FN}(w)$ of free names of w.

▶ **Remark 3.3.** Our notion of α-equivalence on infinite bar strings differs from the equivalence relation generated by relating $x|av$ and $x|bw$ whenever $\langle a\rangle v = \langle b\rangle w$ (as in Definition 3.1 but now for infinite bar strings $v, w \in \overline{\mathbb{A}}_{\mathsf{fs}}^\omega$): The latter equivalence relates two bar strings iff they can be transformed into each other by finitely many α-renamings, while the definition of α-equivalence as per Definition 3.1 allows infinitely many simultaneous α-renamings; e.g. the infinite bar string $(|aa)^\omega = |aa|aa|aa|aa\cdots$ is α-equivalent to $(|aa|bb)^\omega = |aa|bb|aa|bb\cdots$.

▶ **Definition 3.4.** A *literal language*, *bar language* or *data language* is a subset of $\overline{\mathbb{A}}^*$, $\overline{\mathbb{A}}^*/{=_\alpha}$ or $\mathbb{A}^*$, respectively. Similarly, a *literal ω-language*, *bar ω-language* or *data ω-language* is a subset of $\overline{\mathbb{A}}^\omega$, $\overline{\mathbb{A}}^\omega/{=_\alpha}$ or $\mathbb{A}^\omega$, respectively.

▶ **Notation 3.5.** Given a finite or infinite bar string $w \in \overline{\mathbb{A}}^* \cup \overline{\mathbb{A}}^\omega$ we let $\mathsf{ub}(w)$ denote the data word obtained by replacing every occurrence of $|a$ in w by a; e.g. $\mathsf{ub}(|aa|a|bb) = aaabb$. To every bar ($\omega$-)language L we associate the data (ω-)language $D(L)$ given by

$$D(L) = \{\mathsf{ub}(w) : [w]_\alpha \in L\}.$$

▶ **Definition 3.6.** A finite or infinite bar string w is *clean* if for each $a \in \mathsf{FN}(w)$ the letter $|a$ does not occur in w, and for each $a \notin \mathsf{FN}(w)$ the letter $|a$ occurs at most once.

▶ **Lemma 3.7.** *Every bar string $w \in \overline{\mathbb{A}}^* \cup \overline{\mathbb{A}}_{\mathsf{fs}}^\omega$ is α-equivalent to a (not necessarily finitely supported) clean bar string.*

Proof. Since w has finite support, for every occurrence of the letter $|a$ for which a or $|a$ has already occurred before, one can replace $|a$ by $|b$ for some fresh name b and replace the suffix v after $|a$ by $(a\,b)v$. Iterating this yields a clean bar string α-equivalent to w. ◀

▶ **Example 3.8.**

1. The finitely supported infinite bar string $|aa|bb|aa|bb\cdots$ is α-equivalent to the clean bar string $|a_1a_1|a_2a_2|a_3a_3|a_4a_4\cdots$ where the a_i are pairwise distinct names. Note that $|aa|bb|aa|bb\cdots$ is not α-equivalent to any finitely supported clean bar string.
2. For non-finitely supported bar strings the lemma generally fails: if $\mathbb{A} = \{a_1, a_2, a_3, \cdots\}$ then the bar string $a_1|a_1a_2a_3a_4a_5a_6a_7\cdots$ is not α-equivalent to any other bar string, in particular not to a clean one.

For readers familiar with the theory of coalgebras we note that on infinite bar strings with finite support, α-equivalence naturally emerges from a coinductive point of view. Kurz et al. [20] use coinduction to devise a general notion of α-equivalence for infinitary terms over a binding signature, which form the final colgebra for an associated endofunctor on **Nom**. The following is the special case for the endofunctors

$$GX = \overline{\mathbb{A}} \times X \cong \mathbb{A} \times X + \mathbb{A} \times X \qquad \text{and} \qquad FX = \mathbb{A} \times X + [\mathbb{A}]X.$$

Let νF and νG denote their respective final coalgebras. The coalgebra νG is carried by the nominal set $\overline{\mathbb{A}}^{\omega}_{\mathsf{fs}}$ with the usual coalgebra structure $\langle \mathsf{hd}, \mathsf{tl} \rangle \colon \overline{\mathbb{A}}^{\omega}_{\mathsf{fs}} \to \overline{\mathbb{A}} \times \overline{\mathbb{A}}^{\omega}_{\mathsf{fs}}$ decomposing an infinite string into its head and tail. The natural transformation

$$\sigma_X \colon GX \twoheadrightarrow FX \qquad \text{given by} \qquad (a, x) \mapsto (a, x), \quad (\mathsf{I}a, x) \mapsto \langle a \rangle x,$$

then induces a canonical map $e_\alpha \colon \overline{\mathbb{A}}^{\omega}_{\mathsf{fs}} \to \nu F$, viz. the unique homomorphism from the F-coalgebra $\nu G \xrightarrow{\langle \mathsf{hd}, \mathsf{tl} \rangle} G(\nu G) \xrightarrow{\sigma_{\nu G}} F(\nu G)$ into the final coalgebra νF.

▶ **Proposition 3.9.** *For every $v, w \in \overline{\mathbb{A}}^{\omega}_{\mathsf{fs}}$ we have $v =_\alpha w$ if and only if $e_\alpha(v) = e_\alpha(w)$.*

4 Automata over Infinite Bar Strings

We proceed to introduce Büchi RNNAs, our nominal automaton model for bar ω-languages and data ω-languages. It modifies regular nominal nondeterministic automata (RNNAs) [26], originally introduced for bar languages and data languages of finite words, to accept infinite words. Roughly, Büchi RNNAs are to RNNAs what classical Büchi automata [13] are to nondeterministic finite automata. We first recall:

▶ **Definition 4.1** (RNNA [26]).

1. An *RNNA* $A = (Q, R, q_0, F)$ is given by an orbit-finite nominal set Q of *states*, an equivariant relation $R \subseteq Q \times \overline{\mathbb{A}} \times Q$ specifying *transitions*, an *initial state* $q_0 \in Q$ and an equivariant set $F \subseteq Q$ of *final states*. We write $q \xrightarrow{\sigma} q'$ if $(q, \sigma, q') \in R$. The transitions are subject to two conditions:
   (1) *α-invariance*: if $q \xrightarrow{\mathsf{I}a} q'$ and $\langle a \rangle q' = \langle b \rangle q''$, then $q \xrightarrow{\mathsf{I}b} q''$.
   (2) *Finite branching up to α-invariance:* For each $q \in Q$ the sets $\{(a, q') : q \xrightarrow{a} q'\}$ and $\{\langle a \rangle q' : q \xrightarrow{\mathsf{I}a} q'\}$ are finite (equivalently, uniformly finitely supported).
   The *degree* of A, denoted $\mathsf{deg}(A)$, is the maximum of all $|\mathsf{supp}(q)|$ where $q \in Q$.
2. Given a finite bar string $w = \sigma_1 \sigma_2 \cdots \sigma_n \in \overline{\mathbb{A}}^*$ and a state $q \in Q$, a *run* for w from q is a sequence of transitions

   $$q \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} q_n.$$

   The run is *accepting* if q_n is final. The state q *accepts* w if there exists an accepting run for w from q, and the automaton A *accepts* w if its initial state q_0 accepts w. We define

   $$\begin{aligned} L_0(A) &= \{w \in \overline{\mathbb{A}}^* : A \text{ accepts } w\}, && \text{the \textit{literal language accepted by} } A, \\ L_\alpha(A) &= \{[w]_\alpha : w \in \overline{\mathbb{A}}^*,\ A \text{ accepts } w\}, && \text{the \textit{bar language accepted by} } A, \\ D(A) &= D(L_\alpha(A)), && \text{the \textit{data language accepted by} } A. \end{aligned}$$

▶ **Remark 4.2.** Equivalently, an RNNA is an orbit-finite coalgebra

$$Q \longrightarrow 2 \times \mathcal{P}_{\mathsf{ufs}}(\mathbb{A} \times Q) \times \mathcal{P}_{\mathsf{ufs}}([\mathbb{A}]Q)$$

for the functor $FX = 2 \times \mathcal{P}_{\mathsf{ufs}}(\mathbb{A} \times X) \times \mathcal{P}_{\mathsf{ufs}}([\mathbb{A}]X)$ on **Nom**, with an initial state $q_0 \in Q$.

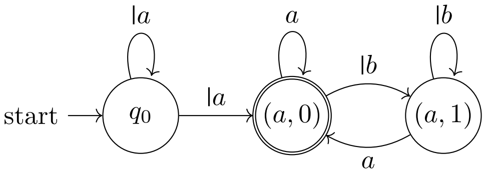
▶ **Definition 4.3** (Büchi RNNA). A *Büchi RNNA* is an RNNA $A = (Q, R, q_0, F)$ used to recognize infinite bar strings as follows. Given $w = \sigma_1 \sigma_2 \sigma_3 \cdots \in \overline{\mathbb{A}}^{\omega}$ and a state $q \in Q$, a *run* for w from q is an infinite sequence of transitions

$$q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} q_2 \xrightarrow{\sigma_3} \cdots .$$

The run is *accepting* if q_n is final for infinitely many $n \in \mathbb{N}$. The state q *accepts* w if there exists an accepting run for w from q, and the automaton A *accepts* w if its initial state q_0 accepts w. We define

$$\begin{aligned} L_{0,\omega}(A) &= \{w \in \overline{\mathbb{A}}^{\omega} : A \text{ accepts } w\}, && \text{the } \textit{literal } \omega\textit{-language accepted by } A,\\ L_{\alpha,\omega}(A) &= \{[w]_\alpha : w \in \overline{\mathbb{A}}^{\omega},\ A \text{ accepts } w\}, && \text{the } \textit{bar } \omega\textit{-language accepted by } A,\\ D_\omega(A) &= D(L_{\alpha,\omega}(A)), && \text{the } \textit{data } \omega\textit{-language accepted by } A. \end{aligned}$$

▶ **Example 4.4.** Consider the Büchi RNNA A with states $\{q_0\} \cup \mathbb{A} \times \{0,1\}$ and transitions as displayed below, where a, b range over distinct names in $\mathbb{A}$. Note that the second node represents the orbit $\mathbb{A} \times \{0\}$ and the third one the orbit $\mathbb{A} \times \{1\}$.



The data ω-language $D_\omega(A)$ consists of all infinite words $w \in \mathbb{A}^\omega$ where some name $a \in \mathbb{A}$ occurs infinitely often.

▶ **Remark 4.5.** For automata consuming infinite words over infinite alphabets, a slightly subtle point is whether to admit arbitrary infinite words as inputs or restrict to finitely supported ones. For the bar language semantics of Büchi RNNAs this choice is inconsequential: we shall see in Proposition 4.7 below that modulo α-equivalence all infinite bar strings accepted by Büchi RNNA are finitely supported. However, for the data language semantics admitting strings with infinite support is important for the decidability of language inclusion, see Remark 6.8.

▶ **Lemma 4.6** ([26, Lem. 5.4]). *Let $A = (Q, R, q_0, F)$ be an RNNA, $q, q' \in Q$ and $a \in \mathbb{A}$.*

1. *If $q \xrightarrow{a} q'$ then $\mathsf{supp}(q') \cup \{a\} \subseteq \mathsf{supp}(q)$.*
2. *If $q \xrightarrow{|a} q'$ then $\mathsf{supp}(q') \subseteq \mathsf{supp}(q) \cup \{a\}$.*
3. *For every bar string $w \in \overline{\mathbb{A}}^* \cup \overline{\mathbb{A}}^\omega$ with a run from q one has $\mathsf{FN}(w) \subseteq \mathsf{supp}(q)$.*

The next proposition will turn out to be crucial in the development that follows. It asserts that, up to α-equivalence, one can always restrict the inputs of a Büchi RNNA to bar strings with a finite number of names, bounded by the degree of the automaton.

▶ **Proposition 4.7.** *Let A be a Büchi RNNA accepting the infinite bar string $w \in \overline{\mathbb{A}}^\omega$. Then it also accepts some $w' \in \overline{\mathbb{A}}^\omega_{\mathsf{fs}}$ such that*

$$w' =_\alpha w \qquad \textit{and} \qquad |\,\mathsf{supp}(q_0) \cup \mathsf{N}(w')| \leq \mathsf{deg}(A) + 1.$$

Proof sketch. Put $m := \mathsf{deg}(A)$, and choose $m+1$ pairwise distinct names $a_1, \ldots, a_{m+1}$ such that $\mathsf{supp}(q_0) \subseteq \{a_1, \ldots, a_{m+1}\}$.

1. One first shows that for every finite bar string $\sigma_1\sigma_2\cdots\sigma_n \in \overline{\mathbb{A}}^*$ and every run

$$q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} q_2 \xrightarrow{\sigma_3} \cdots \xrightarrow{\sigma_n} q_n$$

in A there exists $\sigma_1'\sigma_2'\cdots\sigma_n' \in \overline{\mathbb{A}}^*$ and a run

$$q_0 \xrightarrow{\sigma_1'} q_1' \xrightarrow{\sigma_2'} q_2' \xrightarrow{\sigma_3'} \cdots \xrightarrow{\sigma_n'} q_n' \tag{4.1}$$

such that (1) q_i and q_i' lie in the same orbit for $i = 1, \dots, n$, (2) $\sigma_1'\sigma_2' \cdots \sigma_n' =_\alpha \sigma_1\sigma_2\cdots\sigma_n$, and (3) $\mathsf{N}(\sigma_1'\sigma_2'\cdots\sigma_n') \subseteq \{a_1, \dots, a_{m+1}\}$. The proof is by induction on n and rests on the observation that for every state q at least one of the names $a_1, \dots, a_{m+1}$ is fresh for q.

2. Now suppose that the infinite bar string $w = \sigma_1\sigma_2\sigma_3 \cdots \in \overline{\mathbb{A}}^\omega$ is accepted by A via the accepting run

$$q_0 \xrightarrow{\sigma_1} q_1 \xrightarrow{\sigma_2} q_2 \xrightarrow{\sigma_3} \cdots$$

Consider the set of all partial runs (4.1) satisfying the above conditions (1)–(3). This set organizes into a tree with the edge relation given by extension of runs; its nodes of depth n are exactly the runs (4.1). By part 1 at least one such run exists for each $n \in \mathbb{N}$, i.e. the tree is infinite. It is finitely branching because $\mathsf{supp}(q_i') \subseteq \{a_1, \dots, a_{m+1}\}$ for all i by Lemma 4.6, i.e. there are only finitely many runs (4.1) for each n. Hence, by Kőnig's lemma the tree contains an infinite path. This yields an infinite run

$$q_0 \xrightarrow{\sigma_1'} q_1' \xrightarrow{\sigma_2'} q_2' \xrightarrow{\sigma_3'} \cdots$$

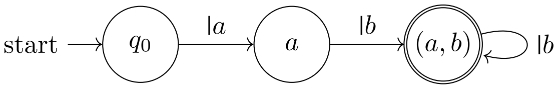
such that q_i' and q_i lie in the same orbit for each i and $\sigma_1' \cdots \sigma_n' =_\alpha \sigma_1 \cdots \sigma_n$ for each n. It follows that the Büchi RNNA A accepts the infinite bar string

$$w' = \sigma_1'\sigma_2'\sigma_3' \cdots .$$

Moreover $w' =_\alpha w$ and $\mathsf{supp}(q_0) \cup \mathsf{N}(w') \subseteq \{a_1, \dots, a_{m+1}\}$ as required. ◀

5 Name-Dropping Modification

Although the transitions of a Büchi RNNA are α-invariant, its literal ω-language generally fails to be closed under α-equivalence. For instance, consider the Büchi RNNA with states $\{q_0\} \cup \mathbb{A} \cup \mathbb{A}^2$ and transitions displayed below, where a, b range over distinct names in $\mathbb{A}$:



The automaton accepts the infinite bar string $|a|b|b|b|b|b\cdots$ for $a \neq b$ but does not accept the α-equivalent $|a|a|a|a|a|a\cdots$: the required transitions $a \xrightarrow{|a} (a, b)$ and $(a, b) \xrightarrow{|a} (a, b)$ do not exist since a is in the least support of the state (a, b), which prevents α-renaming of the given transitions $a \xrightarrow{|b} (a, b)$ and $(a, b) \xrightarrow{|b} (a, b)$. A possible fix is to add a new state "$(\perp, b)$" that essentially behaves like (a, b) but has the name a dropped from its least support.

This idea can be generalized: We will show below how to transform any Büchi RNNA A into a Büchi RNNA $\widetilde{A}$ such that $L_{0,\omega}(\widetilde{A})$ is the closure of $L_{0,\omega}(A)$ under α-equivalence, using the *name-dropping modification* (Definition 5.3). The latter simplifies a construction of the same name previously given for RNNA over finite words [26], and it is the key to the decidability of bar language inclusion proved later on. First, some technical preparations:

▶ **Remark 5.1** (Strong nominal sets)**.** A nominal set Y is called *strong* [28] if for every $\pi \in \mathsf{Perm}(\mathbb{A})$ and $y \in Y$ one has $\pi \cdot y = y$ if and only if $\pi(a) = a$ for all $a \in \mathsf{supp}(y)$. (Note that "if" holds in all nominal sets.) As shown in [24, Lem. 2.4.2] or [23, Cor. B.27(2)] strong nominal sets are up to isomorphism precisely the nominal sets of the form $Y = \sum_{i \in I} \mathbb{A}^{\# X_i}$ where X_i is a finite set and $\mathbb{A}^{\# X}$ denotes the nominal set of all injective maps from X to $\mathbb{A}$, with the group action defined pointwise; $\mathbb{A}^{\# X}$ may be seen as the set of possible configurations of an X-indexed set of registers containing pairwise distinct names. The set of orbits of Y is in bijection with I; in particular, Y is orbit-finite iff I is finite. Strong nominal sets can be regarded as analogues of free algebras in categories of algebraic structures:

1. For every nominal set Z there exists a strong nominal set Y and a surjective equivariant map $e\colon Y \twoheadrightarrow Z$ that is supp-*nondecreasing*, i.e. $\mathsf{supp}(e(y)) = \mathsf{supp}(y)$ for all $y \in Y$ [23, Cor. B.27.1]. (Recall that $\mathsf{supp}(e(y)) \subseteq \mathsf{supp}(y)$ always holds for equivariant maps.) Specifically, one can choose $Y = \sum_{i\in I} \mathbb{A}^{\# n_i}$ where I is the set of orbits of Z and n_i is the size of the support of any element of the orbit i. In particular, if Z is orbit-finite then Y is orbit-finite with the same number of orbits.
2. Strong nominal sets are *projective* w.r.t. supp-nondecreasing quotients [23, Lem. B.28]: given a strong nominal set Y, an equivariant map $h\colon Y \to Z$ and a supp-nondecreasing quotient $e\colon X \twoheadrightarrow Z$, there exists an equivariant map $g\colon Y \to X$ such that $h = e \cdot g$.

▶ **Proposition 5.2.** *For every Büchi RNNA there exists a Büchi RNNA accepting the same literal ω-language whose states form a strong nominal set.*

Proof sketch. Given a Büchi RNNA viewed as a coalgebra

$$Q \xrightarrow{\gamma} 2 \times \mathcal{P}_{\mathsf{ufs}}(\mathbb{A} \times Q) \times \mathcal{P}_{\mathsf{ufs}}([\mathbb{A}]Q),$$

express the nominal set Q of states as a supp-nondecreasing quotient $e\colon P \twoheadrightarrow Q$ of an orbit-finite strong nominal set P using Remark 5.1. Note that the type functor $F(-) = 2 \times \mathcal{P}_{\mathsf{ufs}}(\mathbb{A} \times -) \times \mathcal{P}_{\mathsf{ufs}}([\mathbb{A}](-))$ preserves supp-nondecreasing quotients because all the functors $\mathcal{P}_{\mathsf{ufs}}(-)$, $\mathbb{A} \times -$ and $[\mathbb{A}](-)$ do. Therefore, the right vertical arrow in the square below is supp-nondecreasing, so projectivity of P yields an equivariant map β making it commute:

$$\begin{tikzcd}
P \arrow[r, dashed, "\beta"] \arrow[d, two heads, "e"'] & 2 \times \mathcal{P}_{\mathsf{ufs}}(\mathbb{A} \times P) \times \mathcal{P}_{\mathsf{ufs}}([\mathbb{A}]P) \arrow[d, two heads, "2\times\mathcal{P}_{\mathsf{ufs}}(\mathbb{A}\times e)\times\mathcal{P}_{\mathsf{ufs}}([\mathbb{A}]e)"] \\
Q \arrow[r, "\gamma"] & 2 \times \mathcal{P}_{\mathsf{ufs}}(\mathbb{A} \times Q) \times \mathcal{P}_{\mathsf{ufs}}([\mathbb{A}]Q)
\end{tikzcd}$$

Thus e is a coalgebra homomorphism from (P, β) to (Q, γ). It is not difficult to verify that for every $p \in P$ the states p and $e(p)$ accept the same literal ω-language. In particular, equipping P with an initial state $p_0 \in P$ such that $e(p_0) = q_0$ we see that the Büchi RNNAs P and Q accept the same literal ω-language. ◀

A Büchi RNNA with states $Q = \sum_{i\in I} \mathbb{A}^{\# X_i}$ can be interpreted as an automaton with a finite set I of control states each of which comes equipped with an X_i-indexed set of registers. The following construction shows how to modify such an automaton, preserving the accepted bar ω-language, to become lossy in the sense that after each transition some of the register contents may be nondeterministically erased. Technically, this involves replacing Q with $\widetilde{Q} = \sum_{i\in I} \mathbb{A}^{\$ X_i}$ where $\mathbb{A}^{\$ X_i}$ is a the nominal set of *partial* injective maps from X_i to $\mathbb{A}$, again with the pointwise group action. (Note that while $\mathbb{A}^{\# X_i}$ has only one orbit, $\mathbb{A}^{\$ X_i}$ has one orbit for every subset of X_i.) We represent elements of $\widetilde{Q}$ as pairs (i, r) where $i \in I$ and $r\colon X_i \to \mathbb{A}$ is a partial injective map. The least support of (i, r) is given by

$$\mathsf{supp}(i, r) = \mathsf{supp}(r) = \{r(x) : x \in \mathsf{dom}(r)\},$$

where $\mathsf{dom}(r)$ is the *domain* of r, i.e. the set of all $x \in X_i$ for which $r(x)$ is defined. We say that a partial map $\overline{r}\colon X_i \to \mathbb{A}$ *extends* r if $\mathsf{dom}(r) \subseteq \mathsf{dom}(\overline{r})$ and $r(x) = \overline{r}(x)$ for all $x \in \mathsf{dom}(r)$.

▶ **Definition 5.3** (Name-dropping modification)**.** Let A be a Büchi RNNA whose states form a strong nominal set $Q = \sum_{i\in I} \mathbb{A}^{\# X_i}$ (with I and all X_i finite). The *name-dropping modification* of A is the Büchi RNNA $\widetilde{A}$ defined as follows:

1. The states are given by $\widetilde{Q} = \sum_{i \in I} \mathbb{A}^{\$X_i}$.
2. The initial state of $\widetilde{A}$ is equal to the initial state of A.
3. (i, r) is final in $\widetilde{A}$ iff some (equivalently, every[1]) state $(i, -)$ in A is final.
4. $(i, r) \xrightarrow{a} (j, s)$ in $\widetilde{A}$ iff $\mathsf{supp}(s) \cup \{a\} \subseteq \mathsf{supp}(r)$ and $(i, \overline{r}) \xrightarrow{a} (j, \overline{s})$ in A for some $\overline{r}, \overline{s}$ extending r, s.
5. $(i, r) \xrightarrow{\mathsf{l}a} (j, s)$ in $\widetilde{A}$ iff $\mathsf{supp}(s) \subseteq \mathsf{supp}(r) \cup \{a\}$ and $(i, \overline{r}) \xrightarrow{\mathsf{l}b} (j, \overline{s})$ in A for some $b \# s$ and some $\overline{r}, \overline{s}$ extending $r, (a\, b)s$.

▶ **Theorem 5.4.** *For every Büchi RNNA A, the literal ω-language of $\widetilde{A}$ is the closure of that of A under α-equivalence.*

Proof sketch.

1. One first verifies that the literal ω-language of $\widetilde{A}$ is closed under α-equivalence. To this end, one proves more generally that given α-equivalent infinite bar strings $w = \sigma_1\sigma_2\sigma_3 \cdots$ and $w' = \sigma'_1\sigma'_2\sigma'_3 \cdots$ and a run

   $$(i_0, r_0) \xrightarrow{\sigma_1} (i_1, r_1) \xrightarrow{\sigma_2} (i_2, r_2) \xrightarrow{\sigma_3} \cdots$$

   for w in $\widetilde{A}$ there exists a run of the form

   $$(i_0, r'_0) \xrightarrow{\sigma'_1} (i_1, r'_1) \xrightarrow{\sigma'_2} (i_2, r'_2) \xrightarrow{\sigma'_3} \cdots .$$

   for w' in $\widetilde{A}$. In particular, by definition of the final states of $\widetilde{A}$, the first run is accepting iff the second one is. Similar to the proof of Proposition 4.7, one first establishes the corresponding statement for finite runs by induction on their length, and then extends to the infinite case via Kőnig's lemma.
2. In view of part 1 it remains to prove that A and $\widetilde{A}$ accept the same bar ω-language. Again, this follows from a more general observation: for every infinite run

   $$(i_0, r_0) \xrightarrow{\sigma_1} (i_1, r_1) \xrightarrow{\sigma_2} (i_2, r_2) \xrightarrow{\sigma_3} \cdots$$

   in A there exists an infinite run of the form

   $$(i_0, r'_0) \xrightarrow{\sigma'_1} (i_1, r'_1) \xrightarrow{\sigma'_2} (i_2, r'_2) \xrightarrow{\sigma'_3} \cdots$$

   in $\widetilde{A}$ such that $\sigma'_1\sigma'_2\sigma'_3 \cdots =_\alpha \sigma_1\sigma_2\sigma_3 \cdots$. Moreover, the symmetric statement holds where the roles of A and $\widetilde{A}$ are swapped. As before, one first shows the corresponding statement for finite bar strings and then invokes Kőnig's lemma. ◀

6 Decidability of Inclusion

With the preparations given in the previous sections, we arrive at our main result: language inclusion of Büchi RNNA is decidable, both under bar language semantics and data language semantics. The main ingredients of our proofs below are the name-dropping modification (Theorem 5.4) and the name restriction property stated in Proposition 4.7. We will employ them to show that the language inclusion problems for Büchi RNNA reduce to the inclusion problem for ordinary Büchi automata over finite alphabets. The latter is well-known to be decidable with elementary complexity; in fact, it is PSPACE-complete [18].

[1] This is due to the equivariance of the subset $F \subseteq Q$ of final states.

▶ **Remark 6.1.** For algorithms deciding properties of Büchi RNNAs, a finite representation of the underlying nominal sets of states and transitions is required. A standard representation of a single orbit X is given by picking an arbitrary element $x \in X$ with $\mathsf{supp}(x) = \{a_1, \dots, a_m\}$ and forming the subgroup $G_X \subseteq \mathsf{Perm}(\{a_1, \dots, a_m\})$ of all permutations $\pi \colon \{a_1, \dots, a_m\} \to \{a_1, \dots, a_m\}$ such that $\pi \cdot x = x$. The finite group G_X determines X up to isomorphism [25, Thm. 5.13]. More generally, an orbit-finite nominal set consisting of the orbits $X_1, \dots, X_n$ is represented by a list of n finite permutation groups $G_{X_1}, \dots, G_{X_n}$.

We first consider the bar language semantics:

▶ **Theorem 6.2.** *Bar ω-language inclusion of Büchi RNNAs is decidable in parametrized polynomial space.*

Proof. Let A and B be Büchi RNNAs; the task is to decide whether $L_{\alpha,\omega}(A) \subseteq L_{\alpha,\omega}(B)$. Put $m = \mathsf{deg}(A)$, and choose a set $S \subseteq \mathbb{A}$ of $m+1$ distinct names containing $\mathsf{supp}(q_0)$, where q_0 is the initial state of A. Moreover, form the finite alphabet $\overline{S} = S \cup \{|s : s \in S\}$.

1. We claim that

$$L_{\alpha,\omega}(A) \subseteq L_{\alpha,\omega}(B) \qquad \text{iff} \qquad L_{0,\omega}(A) \cap \overline{S}^{\omega} \subseteq L_{0,\omega}(\widetilde{B}) \cap \overline{S}^{\omega}, \tag{6.1}$$

where $\widetilde{B}$ is the name-dropping modification of B.

($\Rightarrow$) Suppose that $L_{\alpha,w}(A) \subseteq L_{\alpha,\omega}(B)$. Then $L_{0,\omega}(A) \subseteq L_{0,\omega}(\widetilde{B})$ because $L_{0,\omega}(\widetilde{B})$ is the closure of $L_{0,\omega}(B)$ under α-equivalence by Theorem 5.4. In particular, $L_{0,\omega}(A) \cap \overline{S}^{\omega} \subseteq L_{0,\omega}(\widetilde{B}) \cap \overline{S}^{\omega}$.

($\Leftarrow$) Suppose that $L_{0,\omega}(A) \cap \overline{S}^{\omega} \subseteq L_{0,\omega}(\widetilde{B}) \cap \overline{S}^{\omega}$, and let $[w]_\alpha \in L_{\alpha,\omega}(A)$. Thus, $w =_\alpha v$ for some $v \in L_{0,\omega}(A)$. By Proposition 4.7 we know that there exists $v' \in \overline{S}^{\omega}$ accepted by A such that $v' =_\alpha v$. Then

$$v' \in L_{0,\omega}(A) \cap \overline{S}^{\omega} \subseteq L_{0,\omega}(\widetilde{B}) \cap \overline{S}^{\omega} \subseteq L_{0,\omega}(\widetilde{B}).$$

Thus $[w]_\alpha = [v]_\alpha = [v']_\alpha \in L_{\alpha,\omega}(\widetilde{B}) = L_{\alpha,\omega}(B)$, proving $L_{\alpha,\omega}(A) \subseteq L_{\alpha,\omega}(B)$.

2. Now observe that both $L_{0,\omega}(A) \cap \overline{S}^{\omega}$ and $L_{0,\omega}(\widetilde{B}) \cap \overline{S}^{\omega}$ are ω-regular languages over the alphabet $\overline{S}$: they are accepted by the Büchi automata obtained by restricting the Büchi RNNAs A and B, respectively, to the finite set of states with support contained in S and transitions labeled by elements of $\overline{S}$. Inclusion of ω-regular languages is decidable in polynomial space [18]. Let k_A/k_B and m_A/m_B denote the number of orbits and the degree of A/B. Since the support of every state of the Büchi automaton derived from A is contained in the set S and the latter has $(m_A+1)!$ permutations, there are at most

$$k_A \cdot (m_A+1)! \in \mathcal{O}(k_A \cdot 2^{(m_A+1)\log(m_A+1)})$$

states. The Büchi automaton derived from $\widetilde{B}$ has at most

$$k_B \cdot 2^{m_B} \cdot (m_A+1)! \in \mathcal{O}(k_B \cdot 2^{m_B+(m_A+1)\log(m_A+1)})$$

states since $\widetilde{B}$ has at most $k_B \cdot 2^{m_B}$ orbits. Thus, the space required for the inclusion check is polynomial in k_A, k_B and exponential in $m_B + (m_A+1)\log(m_A+1)$. ◀

Next, we turn to the data language semantics.

▶ **Notation 6.3.** Given infinite bar strings $w = \sigma_1\sigma_2\sigma_3\cdots$ and $w' = \sigma_1'\sigma_2'\sigma_3'\cdots$ we write $w \sqsubseteq w'$ if, for all $a \in \mathbb{A}$ and $n \in \mathbb{N}$,

$$\sigma_n = a \quad \text{implies} \quad \sigma_n' \in \{a, |a\} \qquad \text{and} \qquad \sigma_n = |a \quad \text{implies} \quad \sigma_n' = |a.$$

Thus, w' arises from w by arbitrarily putting bars in front of letters in w.

▶ **Lemma 6.4.** *If $v =_\alpha w$ and $v \sqsubseteq v'$, then there exists $w' \in \overline{\mathbb{A}}^\omega$ such that $w \sqsubseteq w'$ and $v' =_\alpha w'$.*

Proof. Let $v' = \sigma'_1\sigma'_2\sigma'_3\cdots$ and $w = \rho_1\rho_2\rho_3\cdots$. We define w' to be the following modification of w: for every $n \in \mathbb{N}$, if $\rho_n = a$ and $\sigma'_n = |b$ for some $a, b \in \mathbb{A}$, replace ρ_n by $|a$. Then $w \sqsubseteq w'$ and $v' =_\alpha w'$ as required. ◀

In the following, we write $D(w)$ for $D(\{[w]_\alpha\})$; thus, $D(w)$ is the set of all $\mathsf{ub}(v) \in \mathbb{A}^\omega$ where $v =_\alpha w$ (cf. Notation 3.5).

▶ **Lemma 6.5.** *Let L be a bar ω-language accepted by some Büchi RNNA and let $w \in \overline{\mathbb{A}}^\omega_{\mathsf{fs}}$. Then $D(w) \subseteq D(L)$ if and only if there exists $w' \sqsupseteq w$ such that $[w']_\alpha \in L$.*

▶ **Corollary 6.6.** *Let K and L be bar ω-languages accepted by Büchi RNNA. Then $D(K) \subseteq D(L)$ if and only if for every $w \in \overline{\mathbb{A}}^\omega_{\mathsf{fs}}$ with $[w]_\alpha \in K$ there exists $w' \sqsupseteq w$ such that $[w']_\alpha \in L$.*

Proof. This follows from Lemma 6.5 using that $D(K) = \bigcup_{[w]_\alpha \in K} D(w)$ and that every α-equivalence class $[w]_\alpha \in K$ has a finitely supported representative by Proposition 4.7. ◀

▶ **Theorem 6.7.** *Data ω-language inclusion of Büchi RNNAs is decidable in parametrized polynomial space.*

Proof. Let A and B be Büchi RNNAs; the task is to decide whether $D_\omega(A) \subseteq D_\omega(B)$. Put $m = \mathsf{deg}(A)$, and choose a set $S \subseteq \mathbb{A}$ of $m+1$ distinct names containing $\mathsf{supp}(q_0)$, where q_0 is the initial state of A. Moreover, form the finite alphabet $\overline{S} = S \cup \{|s : s \in S\}$.

1. For every language $L \subseteq \overline{S}^\omega$ let $L{\downarrow}$ denote the downward closure of L with respect to $\sqsubseteq$:

   $$L{\downarrow} = \{v \in \overline{S}^\omega : \text{there exists } w \in L \text{ such that } v \sqsubseteq w\}.$$

   Every Büchi automaton accepting L can be turned into a Büchi automaton accepting $L{\downarrow}$ by adding the transition $q \xrightarrow{a} q'$ for every transition $q \xrightarrow{|a} q'$ where $a \in S$.
2. We claim that

   $$D_\omega(A) \subseteq D_\omega(B) \qquad \text{iff} \qquad L_{0,\omega}(A) \cap \overline{S}^\omega \subseteq (L_{0,\omega}(\widetilde{B}) \cap \overline{S}^\omega){\downarrow}, \tag{6.2}$$

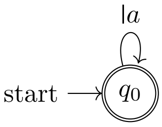
   where $\widetilde{B}$ is the name-dropping modification of B.

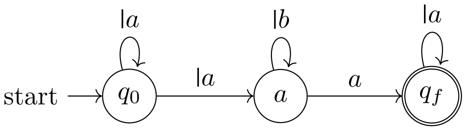
   ($\Rightarrow$) Suppose that $D_\omega(A) \subseteq D_\omega(B)$, that is $D(L_{\alpha,w}(A)) \subseteq D(L_{\alpha,\omega}(B))$, and let $w \in L_{0,\omega}(A) \cap \overline{S}^\omega$. Then $[w]_\alpha \in L_{\alpha,\omega}(A)$, hence by Corollary 6.6 there exists $w' \sqsupseteq w$ such that $[w']_\alpha \in L_{\alpha,\omega}(B)$. Then $w' \in L_{0,\omega}(\widetilde{B}) \cap \overline{S}^\omega$ because $L_{0,\omega}(\widetilde{B})$ is the closure of $L_{0,\omega}(B)$ under α-equivalence by Theorem 5.4. Thus $w \in (L_{0,\omega}(\widetilde{B}) \cap \overline{S}^\omega){\downarrow}$, which proves $L_{0,\omega}(A) \cap \overline{S}^\omega \subseteq (L_{0,\omega}(\widetilde{B}) \cap \overline{S}^\omega){\downarrow}$.

   ($\Leftarrow$) Suppose that $L_{0,\omega}(A) \cap \overline{S}^\omega \subseteq (L_{0,\omega}(\widetilde{B}) \cap \overline{S}^\omega){\downarrow}$. To prove $D_\omega(A) \subseteq D_\omega(B)$, that is $D(L_{\alpha,\omega}(A)) \subseteq D(L_{\alpha,\omega}(B))$, we use Corollary 6.6. Thus, let $[w]_\alpha \in L_{\alpha,\omega}(A)$. Then $w =_\alpha v$ for some $v \in L_{0,\omega}(A)$. By Proposition 4.7 we may assume that $v \in \overline{S}^\omega$. By hypothesis this implies $v \in (L_{0,\omega}(\widetilde{B}) \cap \overline{S}^\omega){\downarrow}$, that is, there exists $v' \sqsupseteq v$ such that $v' \in L_{0,\omega}(\widetilde{B})$. By Lemma 6.4, there exists $w' \sqsupseteq w$ such that $w' =_\alpha v'$. Then $[w']_\alpha = [v']_\alpha \in L_{\alpha,\omega}(\widetilde{B}) = L_{\alpha,\omega}(B)$, as required.
3. The decidability of $D_\omega(A) \subseteq D_\omega(B)$ now follows from part 1 and (6.2) in complete analogy to the proof of Theorem 6.2. ◀

▶ **Remark 6.8.** In the above theorem it is crucial to admit non-finitely supported data words: it is an open problem whether the inclusion $D_\omega(A) \cap \mathbb{A}^\omega_{\mathsf{fs}} \subseteq D_\omega(B) \cap \mathbb{A}^\omega_{\mathsf{fs}}$ is decidable. In fact, our decidability proof relies on Corollary 6.6 as a key ingredient, and the latter fails if the condition $D(K) \subseteq D(L)$ is replaced by the weaker condition $D(K) \cap \mathbb{A}^\omega_{\mathsf{fs}} \subseteq D(L) \cap \mathbb{A}^\omega_{\mathsf{fs}}$.

To see this, let K and L be the bar ω-languages accepted by the two Büchi RNNAs displayed below, where a, b range over names in $\mathbb{A}$ and $a \neq b$:





We have $D(K) = \mathbb{A}^\omega$ and $D(L)$ consists of all data words in $\mathbb{A}^\omega$ in which some name occurs at least twice. Thus, $D(K) \cap \mathbb{A}^\omega_{\mathsf{fs}} = D(L) \cap \mathbb{A}^\omega_{\mathsf{fs}} = \mathbb{A}^\omega_{\mathsf{fs}}$, in particular the inclusion $D(K) \cap \mathbb{A}^\omega_{\mathsf{fs}} \subseteq D(L) \cap \mathbb{A}^\omega_{\mathsf{fs}}$ holds. Consider the infinite bar string $w = |a|a|a \cdots$ and note that $w' \sqsupseteq w$ implies $w' = w$. Then $[w]_\alpha \in K$ but $[w]_\alpha \notin L$ since every bar string accepted by the right-hand automaton contains some letter from $\mathbb{A}$. Thus, "only if" fails in Corollary 6.6.

7 Relation to Other Automata Models

We conclude this paper by comparing Büchi RNNA with two related automata models over infinite words. Ciancia and Sammartino [5] consider deterministic nominal automata accepting data ω-languages $L \subseteq \mathbb{A}^\omega$ via a Muller acceptance condition. More precisely, a *nominal deterministic Muller automaton* (*nDMA*) $A = (Q, \delta, q_0, \mathcal{F})$ is given by an orbit-finite nominal set Q of states, and equivariant map $\delta \colon Q \times \mathbb{A} \to Q$ representing transitions, an initial state $q_0 \in Q$, and a set $\mathcal{F} \subseteq \mathcal{P}(\mathsf{orb}(Q))$ where $\mathsf{orb}(Q)$ is the finite set of orbits of Q. Every input word $w = a_1 a_2 a_3 \cdots \in \mathbb{A}^\omega$ has a unique run $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots$ where $q_{i+1} = \delta(q_i, a_{i+1})$ for $i = 0, 1, 2, \ldots$. The word w is *accepted* if the set $\{O \in \mathsf{orb}(Q) : q_n \in O \text{ for infinitely many } n\}$ lies in $\mathcal{F}$. The data ω-language *accepted* by the automaton is the set of all words $w \in \mathbb{A}^\omega$ whose run is accepting.

As for Büchi RNNA, language inclusion is decidable for nDMA [5, Thm. 4]. In terms of expressive power the two models are incomparable, as witnessed by the data ω-languages

$$K = \{w \in \mathbb{A}^\omega : \text{some } a \in \mathbb{A} \text{ occurs infinitely often in } w\},$$
$$L = \{w \in \mathbb{A}^\omega : w \text{ does not have the suffix } a^\omega \text{ for any } a \in \mathbb{A}\}.$$

▶ **Proposition 7.1.**

1. *The language K is accepted by a Büchi RNNA but not by any nDMA.*
2. *The language L is accepted by an nDMA but not by any Büchi RNNA.*

Proof.

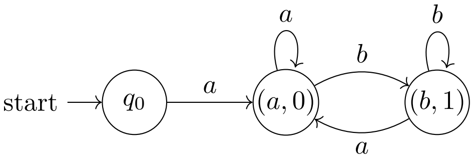
1. We have seen in Example 4.4 that the language K is accepted by a Büchi RNNA. We claim that K is not accepted by any nDMA. Since the class of languages accepted by nDMA is closed under complement, it suffices to show that the language

$$\overline{K} = \{w \in \mathbb{A}^\omega : \text{each } a \in \mathbb{A} \text{ occurs only finitely often in } w\}$$

is not accepted by any nDMA. Suppose towards a contradiction that $A = (Q, \delta, q_0, \mathcal{F})$ is an nDMA accepting $\overline{K}$. Let m be the maximum of all $|\mathsf{supp}(q)|$ where $q \in Q$. Fix $m+1$ pairwise distinct names $a_1, \ldots, a_{m+1} \in \mathbb{A}$ and an arbitrary word $w_0 \in \overline{K}$.
Choose a factorization $w_0 = v_1 w_1$ ($v_1 \in \mathbb{A}^*, w_1 \in \mathbb{A}^\omega$) such that all occurrences of $a_1, \ldots, a_{m+1}$ in w_0 lie in the finite prefix v_1. Let q_1 be the state reached from q_0 on input v_1. Then $\mathsf{supp}(q_1)$ does not contain all of the names $a_1, \ldots, a_{m+1}$, say $a_i \mathrel{\#} q_1$. Choose any name $a \in \mathbb{A}$ occurring in w_1 such that $a \# q_1$. Then $q_1 = (a\, a_i) q_1$ accepts $(a\, a_i) w_1$ since by equivariance the run of $(a\, a_i) w_1$ from q_1 visits the same orbits as the run of w_1.
Now repeat the same process with q_1 and $(a\, a_i) w_1$ in lieu of q_0 and w_0, and let $(a\, a_i) w_1 = v_2 w_2$ denote the corresponding factorization; note that v_2 is nonempty because $(a\, a_i) w_1$ contains the name a_i. Continuing in this fashion yields an infinite word

$v = v_1v_2v_3\ldots$ such that each v_i ($i > 1$) contains at least one of the letters $a_1, \ldots, a_{m+1}$, and the run of v traverses the same orbits (in the same order) as the run of w. Thus v is accepted by the nDMA A although $v \notin \overline{K}$, a contradiction.

2. The language L is accepted by the nDMA with states $\{q_0\} \cup \mathbb{A} \times \{0,1\}$ and transitions as displayed below, where a, b range over distinct names in $\mathbb{A}$. The acceptance condition is given by $\mathcal{F} = \{\{\mathbb{A} \times \{0\}, \mathbb{A} \times \{1\}\}\}$.



We claim that L is not accepted by any Büchi RNNA. Suppose to the contrary that $A = (Q, R, q_0, F)$ is a Büchi RNNA with $D_\omega(A) = L$. By Theorem 5.4 we may assume that $L_{0,\omega}(A)$ is closed under α-equivalence. Fix an arbitrary word $w = a_1a_2a_3\cdots$ whose names are pairwise distinct and not contained in $\mathsf{supp}(q_0)$. Then $w \in L$, so there exists $v \in L_{0,\omega}(A)$ such that $\mathsf{ub}(v) = w$. We claim that $v = |a_1|a_2|a_3\cdots$. Indeed, if the nth letter of v is a_n, then in an accepting run for v in A the state q reached before reading a_n must have $a_n \in \mathsf{supp}(q)$ by Lemma 4.6.3. But this is impossible because $\mathsf{supp}(q) \subseteq \mathsf{supp}(q_0) \cup \{a_1, \ldots, a_{n-1}\}$ again by Lemma 4.6.

Thus $v =_\alpha |a|a|a\cdots$. This implies $a^\omega \in D_\omega(A)$ although $a^\omega \notin L$, a contradiction. ◀

Let us note that the above result does not originate in weakness of the Büchi acceptance condition. One may generalize Büchi RNNA to *Muller RNNA* where the final states $F \subseteq Q$ are replaced by a set $\mathcal{F} \subseteq \mathcal{P}(\mathsf{orb}(Q))$, and a bar string $w \in \overline{\mathbb{A}}^\omega$ is said to be *accepted* if there exists a run for w such that the set of orbits visited infinitely often lies in $\mathcal{F}$. However, as for classical nondeterministic Büchi and Muller automata, this does not affect expressivity:

▶ **Proposition 7.2.** *A literal ω-language is accepted by a Büchi RNNA if and only if it is accepted by a Muller RNNA.*

Finally, we mention a tight connection between Büchi RNNA and *session automata* [4]. The data (ω-)language associated to a (Büchi) RNNA uses a *local freshness* semantics in the sense that its definition considers possibly non-clean bar strings accepted by A. In some applications, e.g. nonce generation, a more suitable semantics is given by *global freshness* where only clean bar strings are admitted, i.e. one associates to A the data languages

$$D_{\#}(A) = \{\mathsf{ub}(w) : w \in \overline{\mathbb{A}}^* \text{ clean and } [w]_\alpha \in L_\alpha(A)\},$$
$$D_{\omega,\#}(A) = \{\mathsf{ub}(w) : w \in \overline{\mathbb{A}}^\omega \text{ clean and } [w]_\alpha \in L_{\alpha,\omega}(A)\}.$$

For instance, for the Büchi RNNA A from Example 4.4 the language $D_{\omega,\#}(A)$ consists of all infinite words $w \in \mathbb{A}^\omega$ where exactly one name $a \in \mathbb{A}$ occurs infinitely often and every name $b \neq a$ occurs at most once.

Under global freshness semantics, it has been observed in previous work [26] that a data language $L \subseteq \mathbb{A}^*$ is accepted by some session automaton iff $L = D_{\#}(A)$ for some RNNA A whose initial state q_0 has empty support. An analogous correspondence holds for Büchi RNNA and data ω-languages $L \subseteq \mathbb{A}^\omega$ if the original notion of session automata is generalized to infinite words with a Büchi acceptance condition.

8 Conclusions and Future Work

We have introduced *Büchi regular nondeterministic nominal automata* (Büchi RNNAs), an automaton model for languages of infinite words over infinite alphabets. Büchi RNNAs allow for inclusion checking in elementary complexity (parametrized polynomial space) despite the fact that they feature full nondeterminism and do not restrict the number of registers (contrastingly, even for register automata over finite words [16], inclusion checking becomes decidable only if the number of registers is bounded to at most 2).

An important further step will be to establish a logic-automata correspondence of Büchi RNNAs with a suitable form of linear temporal logic on infinite data words.

A natural direction for generalization is to investigate RNNAs over infinite trees with binders, modeled as coalgebras of type $\mathcal{P}_{\mathsf{ufs}}F$ for a functor F associated to a binding signature and equipped with the ensuing notion of α-equivalence due to Kurz et al. [20].

Finally, we would like to explore the bar language and data language semantics of Büchi RNNA from the perspective of coalgebraic trace semantics [29], where infinite behaviours emerge as solutions of (nested) fixed point equations in Kleisli categories.

References

1 Hendrik Pieter Barendregt. *The lambda calculus - its syntax and semantics*, volume 103 of *Studies in logic and the foundations of mathematics*. North-Holland, 1985.

2 Mikołaj Bojańczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. Comput. Log.*, 12(4):27:1–27:26, 2011. doi:10.1145/1970398.1970403.

3 Mikołaj Bojańczyk, Bartek Klin, and Sławomir Lasota. Automata theory in nominal sets. *Log. Methods Comput. Sci.*, 10(3), 2014. doi:10.2168/LMCS-10(3:4)2014.

4 Benedikt Bollig, Peter Habermehl, Martin Leucker, and Benjamin Monmege. A robust class of data languages and an application to learning. *Log. Meth. Comput. Sci.*, 10(4), 2014. doi:10.2168/LMCS-10(4:19)2014.

5 Vincenzo Ciancia and Matteo Sammartino. A class of automata for the verification of infinite, resource-allocating behaviours. In *Trustworthy Global Computing, TGC 2014*, volume 8902 of *LNCS*, pages 97–111. Springer, 2014. doi:10.1007/978-3-662-45917-1_7.

6 Thomas Colcombet and Amaldev Manuel. Generalized data automata and fixpoint logic. In *Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014*, volume 29 of *LIPIcs*, pages 267–278. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.267.

7 Wojciech Czerwiński, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. *J. ACM*, 68(1):7:1–7:28, 2021. doi:10.1145/3422822.

8 Wojciech Czerwiński and Lukasz Orlikowski. Reachability in vector addition systems is Ackermann-complete. *CoRR*, 2021. arXiv:2104.13866.

9 Stéphane Demri and Ranko Lazić. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009. doi:10.1145/1507244.1507246.

10 Murdoch J. Gabbay. Foundations of nominal techniques: logic and semantics of variables in abstract syntax. *Bull. Symb. Log.*, 17(2):161–229, 2011. doi:10.2178/bsl/1305810911.

11 Murdoch J. Gabbay and Vincenzo Ciancia. Freshness and name-restriction in sets of traces with names. In *Foundations of Software Science and Computation Structures, FOSSACS 2011*, volume 6604 of *LNCS*, pages 365–380. Springer, 2011. doi:10.1007/978-3-642-19805-2.

12 Murdoch J. Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *Logic in Computer Science, LICS 1999*, pages 214–224. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782617.

13 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002. doi:10.1007/3-540-36387-4.

14 Radu Grigore, Dino Distefano, Rasmus Petersen, and Nikos Tzevelekos. Runtime verification based on register automata. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2013*, volume 7795 of *LNCS*, pages 260–276. Springer, 2013. doi:10.1007/978-3-642-36742-7_19.

15 Orna Grumberg, Orna Kupferman, and Sarai Sheinvald. Variable automata over infinite alphabets. In *Language and Automata Theory and Applications, LATA 2010*, volume 6031 of *LNCS*, pages 561–572. Springer, 2010. doi:10.1007/978-3-642-13089-2.

16 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994. doi:10.1016/0304-3975(94)90242-9.

17 Ahmet Kara, Thomas Schwentick, and Tony Tan. Feasible automata for two-variable logic with successor on data words. In *Language and Automata Theory and Applications, LATA 2012*, volume 7183 of *LNCS*, pages 351–362. Springer, 2012. doi:10.1007/978-3-642-28332-1_30.

18 Orna Kupferman and Moshe Y. Vardi. Verification of fair transisiton systems. In *Computer Aided Verification, CAV 1996*, volume 1102 of *LNCS*, pages 372–382. Springer, 1996. doi:10.1007/3-540-61474-5_84.

19 Klaas Kürtz, Ralf Küsters, and Thomas Wilke. Selecting theories and nonce generation for recursive protocols. In *Formal methods in security engineering, FMSE 2007*, pages 61–70. ACM, 2007. doi:10.1145/1314436.1314445.

20 Alexander Kurz, Daniela Petrisan, Paula Severi, and Fer-Jan de Vries. Nominal coalgebraic data types with applications to lambda calculus. *Log. Methods Comput. Sci.*, 9(4), 2013. doi:10.2168/LMCS-9(4:20)2013.

21 Ranko Lazić. Safely freezing LTL. In *Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2006*, volume 4337 of *LNCS*, pages 381–392. Springer, 2006. doi:10.1007/11944836_35.

22 Jérôme Leroux. The reachability problem for Petri nets is not primitive recursive. *CoRR*, 2021. arXiv:2104.12695.

23 Stefan Milius and Henning Urbat. Equational axiomatization of algebras with structure. In *Foundations of Software Science and Computation Structures, FOSSACS 2019*, volume 11425 of *LNCS*, pages 400–417. Springer, 2019. Full version available at arXiv:1812.02016. doi:10.1007/978-3-030-17127-8_23.

24 Daniela Petrişan. *Investigations into Algebra and Topology over Nominal Sets*. PhD thesis, University of Leicester, 2011.

25 Andrew Pitts. *Nominal Sets: Names and Symmetry in Computer Science*. Cambridge University Press, 2013.

26 Lutz Schröder, Dexter Kozen, Stefan Milius, and Thorsten Wißmann. Nominal automata with name binding. In *Foundations of Software Science and Computation Structures, FOSSACS 2017*, volume 10203 of *LNCS*, pages 124–142, 2017. doi:10.1007/978-3-662-54458-7_8.

27 Christoph Stockhusen and Till Tantau. Completeness results for parameterized space classes. In *Parameterized and Exact Computation, IPEC 2013*, volume 8246 of *LNCS*, pages 335–347. Springer, 2013. doi:10.1007/978-3-319-03898-8.

28 Nikos Tzevelekos. *Nominal Game Semantics*. PhD thesis, University of Oxford, 2008.

29 Natsuki Urabe, Shunsuke Shimizu, and Ichiro Hasuo. Coalgebraic trace semantics for buechi and parity automata. In *Concurrency Theory, CONCUR 2016*, volume 59 of *LIPIcs*, pages 24:1–24:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.24.

Enforcing ω-Regular Properties in Markov Chains by Restarting

Javier Esparza
Technische Universität München, Germany

Stefan Kiefer
University of Oxford, UK

Jan Křetínský
Technische Universität München, Germany

Maximilian Weininger
Technische Universität München, Germany

Abstract

Restarts are used in many computer systems to improve performance. Examples include reloading a webpage, reissuing a request, or restarting a randomized search. The design of restart strategies has been extensively studied by the performance evaluation community. In this paper, we address the problem of designing universal restart strategies, valid for arbitrary finite-state Markov chains, that enforce a given ω-regular property while not knowing the chain. A strategy enforces a property φ if, with probability 1, the number of restarts is finite, and the run of the Markov chain after the last restart satisfies φ. We design a simple "cautious" strategy that solves the problem, and a more sophisticated "bold" strategy with an almost optimal number of restarts.

2012 ACM Subject Classification Theory of computation → Verification by model checking

Keywords and phrases Markov chains, omega-regular properties, runtime enforcement

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.5

Funding This work was supported by the Research Training Group GRK 2428 CONVEY of the German Research Council (DFG) and by the ERC project PaVeS (Advanced Grant 787367).
Stefan Kiefer: supported by a Royal Society University Research Fellowship.

1 Introduction

Many computing systems offer the possibility to *restart* a computation or an interaction that is suspected to have failed, in order to improve performance. The standard example is the reload button of a web browser: when a download seems to "hang", pressing the button can lead to a faster, even to an immediate download. Similar situations appear in networks, where resending a message can lead to a faster acknowledgment, in randomized search, where the search can be restarted with a new seed, or in software rejuvenation, where performing a garbage collection, flushing caches, or simply rebooting may improve performance (see e.g. [7, 10, 12, 13, 20, 22]). Performance evaluation has extensively explored how to find optimal restart strategies in stochastic timed systems, usually under strong assumptions on the distribution of the times at which events occur.

In verification, *liveness properties* are abstractions of performance requirements: "every requested webpage will eventually be downloaded" is an abstraction of "every requested webpage will be downloaded within at most 3 seconds", or of some more complicated statement. The advantage is that they can be checked even when timing information is not available. However, to the best of our knowledge restart strategies for liveness properties have not been studied in the probabilistic, untimed setting. In this paper we study this question.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 5; pp. 5:1–5:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Consider a liveness property φ like "every requested webpage will eventually be downloaded". Assume that the system is modelled by a finite-state Markov chain, including faulty behaviour of the TCP protocol, or the congestion phenomena causing requests to hang. We only know the initial state of the chain, and we can execute a probabilistic program that, given a state of the chain, returns a successor state according to the transition probabilities; we do not have access to the code of this program, or it could be very complex, and so we do not know the probabilities. We can monitor runs of the chain and record the sequence of states they visit; further, we are allowed to restart the system at any moment. The problem is to design a *universal* restart strategy, valid for every chain, satisfying the following two properties:

(1) With probability 1, the number of restarts is finite, and the run of the Markov chain after the last restart satisfies φ.[1]

(2) The expected number of restarts R, and the expected number of steps S to a restart (conditioned on the occurrence of the restart) are not too high (we make this more precise later).

We say that strategies satisfying (1) *enforce* φ. If φ has zero probability, then no strategy can enforce φ, and so we assume that φ has non-zero probability. Under this assumption, it is easy to design naive enforcing strategies for safety and co-safety properties. For a safety property like $\mathbf{G}p$ it suffices to restart whenever the current state does not satisfy p; indeed, since $\mathbf{G}p$ has positive probability by assumption, eventually the chain executes a run satisfying $\mathbf{G}p$ almost surely, and this run is not aborted. Similarly, for $\mathbf{F}p$ we can abort the first execution after one step, the second after two steps etc., until a state satisfying p is reached. Since $\mathbf{F}p$ has positive probability by assumption, at least one reachable state satisfies p, and we will almost surely visit it. However, these naive strategies lead to far too many restarts on average. Further, for reactivity properties like "every requested webpage will eventually be downloaded" even the problem of finding any enforcing strategy is already challenging: Unlike for $\mathbf{G}p$ and $\mathbf{F}p$, the strategy can never be sure that every extension of the current execution will satisfy the property, or that every extension will violate it.

Our first result shows that, perhaps surprisingly, ideas introduced in [6] on the detection of strongly connected components at runtime lead to a very simple enforcing strategy. Let $\mathcal{M}$ be the (unknown) Markov chain of the system, and let $\mathcal{A}$ be a deterministic Rabin automaton for φ. Say that a run of the product chain $\mathcal{M} \times \mathcal{A}$ is *good* if it satisfies φ, and *bad* otherwise. We define a set of *suspect* finite executions satisfying two properties:

(a) bad runs almost surely have a suspect prefix; and

(b) if the set of good runs has nonzero probability, then the set of runs without suspect prefixes also has nonzero probability.

The strategy restarts the chain whenever the current execution is suspect. We call it the *cautious strategy*, or, since it must be implemented by monitoring the system, the *cautious monitor*. By property (a) the cautious monitor aborts bad runs almost surely, and by property (b) almost surely the system eventually executes a run without suspect prefixes, which by (a) is necessarily good.

While the cautious monitor is very simple, it does not satisfy condition (2): in the worst case, both R and S are exponential in the number of states of the chain. A simple analysis shows that, without further information on the chain, the exponential dependence of S on the number of states is unavoidable. However, the exponential dependence of R on the number

[1] More precisely, the property is enforced only under the assumption that φ has non-zero probability, otherwise there is no such strategy.

of states can be avoided: using a technique of [6], we define a *bold monitor* for which the expected number of restarts is almost optimal. Observe that if the property φ has probability p_φ, then the number of restarts needed by *any* monitor, even those that know the chain, is $1/\mathsf{p}_\varphi$, because, loosely speaking, that is the expected number of runs until the chain executes a run satisfying φ. Our bold monitor achieves $c + 1/(\mathsf{p}_\varphi(1-\epsilon))$ restarts on average for any given $\epsilon > 0$, where c is a constant independent of both the chain and φ.

We also develop an efficient data structure that the monitor may use to keep track of the relevant parameters of the current run. Finally, we illustrate the behaviour of our monitors on models from the standard PRISM Benchmark Suite [15], and compare our theoretical bounds to the experimental values.

Related work. Our work is related to runtime enforcement [17]. In this approach a property is enforced by automatically constructing a monitor that inspects the execution online in an incremental way, and takes action whenever it violates the property; only safety properties are enforceable. The ideas of [17] have been extended to some non-safety properties and timed properties (see e.g. [3, 16, 8, 9]). To the best of our knowledge, restarts as runtime enforcers in a probabilistic setting have not been considered. Runtime monitoring of stochastic systems modelled as Hidden Markov Chains (HMM) has been studied by Sistla et al. in a number of papers [18, 11, 19], but these papers concentrate on monitoring violations of a property, not on enforcing it.

Our problem is also related to learning Markov chains, e.g., [5, 4, 21, 1]. A simple and correct restart strategy based on learning is to store *all* the states and transitions of the chain $\mathcal{M} \times \mathcal{A}$ visited so far, yielding a *currently explored graph*, and restart whenever the current state belongs to a non-accepting and non-trivial bottom strongly connected component of this graph. However, the memory consumption of such a strategy is very high. We focus on strategies that only store (part of) the current run, at lower memory cost. Moreover, many real systems exhibit a “fat but shallow” topology, i.e., a large ratio between the number of paths and their length [6]. In such chains, states are rarely revisited, and so storing all past states for eventual use in future runs is inefficient. Our strategies only need a few restarts; more precisely, the number of restarts only depends on the inverse of the probability of φ, but not on the size of the chain. These points are discussed in more detail in Remark 5 and in Section 4.3.

2 Preliminaries and running example

Directed graphs. A directed graph is a pair $G = (V, E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. A path (infinite path) of G is a finite (infinite) sequence $\pi = v_0, v_1 \ldots$ of vertices such that $(v_i, v_{i+1}) \in E$ for every $i = 0, 1 \ldots$. We denote the empty path by λ and concatenation of paths π_1 and π_2 by $\pi_1 \,.\, \pi_2$. A graph G is strongly connected if for every two vertices v, v' there is a path leading from v to v'. A graph $G' = (V', E')$ is a subgraph of G, denoted $G' \preceq G$, if $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$; we write $G' \prec G$ if $G' \preceq G$ and $G' \neq G$. A graph $G' \preceq G$ is a strongly connected component (SCC) of G if it is strongly connected and no graph G'' satisfying $G' \prec G'' \preceq G$ is strongly connected. An SCC $G' = (V', E')$ of G is a bottom SCC (BSCC) if $v \in V'$ and $(v, v') \in E$ imply $v' \in V'$.

Markov chains. A *Markov chain (MC)* is a tuple $\mathcal{M} = (S, \mathbf{P}, \mu)$, where S is a finite set of states, $\mathbf{P} : S \times S \to [0, 1]$ is the transition probability matrix, such that for every $s \in S$ it holds $\sum_{s' \in S} \mathbf{P}(s, s') = 1$, and μ is a probability distribution over S. The graph of $\mathcal{M}$ has S

as vertices and $\{(s,s') \mid \mathbf{P}(s,s')>0\}$ as edges. Abusing language, we also use $\mathcal{M}$ to denote the graph of $\mathcal{M}$. We let $\mathsf{p_{min}} := \min(\{\mathbf{P}(s,s')>0 \mid s,s' \in S\})$ denote the smallest positive transition probability in $\mathcal{M}$. A *run* of $\mathcal{M}$ is an infinite path $\rho = s_0 s_1 \cdots$ of $\mathcal{M}$; we let $\rho[i]$ denote the state s_i. Each (finite) path π in $\mathcal{M}$ determines the set of runs $\mathsf{Cone}(\pi)$ consisting of all runs that start with π. To $\mathcal{M}$ we assign the probability space $(\mathsf{Runs}, \mathcal{F}, \mathbb{P})$, where Runs is the set of all runs in $\mathcal{M}$, $\mathcal{F}$ is the σ-algebra generated by all $\mathsf{Cone}(\pi)$, and $\mathbb{P}$ is the unique probability measure such that $\mathbb{P}[\mathsf{Cone}(s_0 s_1 \cdots s_k)] = \mu(s_0) \cdot \prod_{i=1}^{k} \mathbf{P}(s_{i-1}, s_i)$, where the empty product equals 1. The expected value of a random variable $f : \mathsf{Runs} \to \mathbb{R}$ is $\mathbb{E}[f] = \int_{\mathsf{Runs}} f \, d\mathbb{P}$.

Given a finite set Ap of atomic propositions, a *labelled Markov chain* (LMC) is a tuple $\mathcal{M} = (S, \mathbf{P}, \mu, Ap, L)$, where $(S, \mathbf{P}, \mu)$ is a MC and $L : S \to 2^{Ap}$ is a labelling function. Given a labelled Markov chain $\mathcal{M}$ and an ω-regular property φ, we are interested in the measure $\mathbb{P}[\mathcal{M} \models \varphi] := \mathbb{P}[\{\rho \in \mathsf{Runs} \mid L(\rho) \models \varphi\}]$, where L is naturally extended to runs by $L(\rho)[i] = L(\rho[i])$ for all i.

In this work, we assume that $\mathcal{M}$, and in particular S, are unknown to the algorithms; when a simulation run is observed, we can distinguish whether the current state has already been seen and at which step, but we cannot match the state with any concrete element of S even if we knew $\mathcal{M}$.

Deterministic Rabin Automata and product Markov chain. For every ω-regular property (in particular, for every LTL property) φ there is a *deterministic Rabin automaton* (DRA) $\mathcal{A} = (Q, 2^{Ap}, \gamma, q_o, Acc)$ that accepts all runs that satisfy φ [2]. Here Q is a finite set of states, $\gamma : Q \times 2^{Ap} \to Q$ is the transition function, $q_o \in Q$ is the initial state, and $Acc \subseteq 2^Q \times 2^Q$ is the acceptance condition.

The product of a MC $\mathcal{M}$ and DRA $\mathcal{A}$ is the Markov chain $\mathcal{M} \otimes \mathcal{A} = (S \times Q, \mathbf{P}', \mu')$, where $\mathbf{P}'((s,q),(s',q')) = \mathbf{P}(s,s')$ if $q' = \gamma(q, L(s'))$ and $\mathbf{P}'((s,q),(s',q')) = 0$ otherwise, and $\mu'(s,q) = \mu(s)$ if $\gamma(q_o, L(s)) = q$ and $\mu'(s,q) = 0$ otherwise. Note that $\mathcal{M} \otimes \mathcal{A}$ has the same smallest transition probability $\mathsf{p_{min}}$ as $\mathcal{M}$.

A run of $\mathcal{M} \otimes \mathcal{A}$ is *good* if it satisfies φ, i.e., if it is accepted by $\mathcal{A}$, and *bad* otherwise. An SCC B of $\mathcal{M} \otimes \mathcal{A}$ is *good* if there exists a Rabin pair $(E,F) \in Acc$ such that $B \cap (S \times E) = \emptyset$ and $B \cap (S \times F) \neq \emptyset$. Otherwise, the SCC is *bad*. Observe that good runs of $\mathcal{M} \otimes \mathcal{A}$ almost surely reach a good BSCC (i.e., more formally, the probability that a run satisfies φ and does not reach a good BSCC is 0), and bad runs almost surely reach a bad BSCC (i.e., more formally, the probability that a run does not satisfy φ and does not reach a bad BSCC is also 0).

▶ **Example 1** (running). We present a strongly idealized but illustrative running example. Consider a concurrent system where multiple threads execute transactions (e.g., the database of an online shop). Every thread first acquires locks on a number of variables (which may dynamically depend on the values of the variables it has already acquired), then executes its task, and releases all locks. A thread may have to repeatedly try to acquire a lock on a given

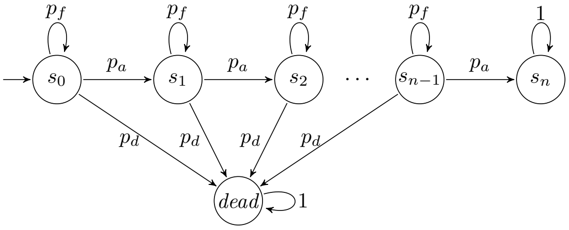


Figure 1 An idealized example.

variable, since it can be currently held by another thread. Moreover a thread that acquires a lock on a variable, say v, reaches a deadlock if the next variable it needs, say v', is currently held by another thread trying to get a lock on v.

The deadlock problem can be solved by maintaining a dependency graph of all transactions, and breaking all cyclic dependencies by restarting at least one of the threads involved. However, this graph is usually too large to be practical. Under the assumption that threads are scheduled stochastically, there are simpler mechanisms requiring no communication between threads: Each thread counts the number of attempts it makes at acquiring the same variable, and restarts (i.e., releases all its locks) when the number becomes "too high". We want to design such a mechanism, assuming that the thread has only the following information (or, equivalently, assumes that the following hypotheses are correct): (1) its interaction with the rest of the system can be modelled by a discrete finite-state Markov chain, whose states store the sequence of locks acquired by the thread so far; (2) the probability that an attempt to acquire a lock fails, and that it succeeds but leads to a deadlock only depend on the current state, not on the number of attempts; and (3) the probability of the runs of the chain in which the thread eventually acquires all locks it needs is positive. The thread does not know any probability, or even the number of locks it needs. At every moment in time, the thread can choose between trying to acquire the next lock, or restarting and releasing all locks. The challenge is to design a restarting strategy ensuring that the thread will execute its task with probability 1 while keeping the number of resets low. Observe that, even though the thread does not know the chain, it can tell at each step whether it stays in the same state, or moves to a new state it has not yet visited. Indeed, an attempt to acquire a lock either fails, in which case the thread stays in the same state, or succeeds, in which case it moves to a new state not visited so far, because the set of acquired locks is larger. In particular, the thread can maintain a list $k_0, k_1, k_2, \ldots$ indicating the number k_i of visits to the i-th state. However, the thread does not know if the current state is a deadlock or not.

Consider a simple case in which the thread always needs locks on the same n variables, the probability of acquiring the lock and not reaching a deadlock afterwards, acquiring the lock and reaching a deadlock, and failing to acquire the lock are p_a, p_d and $p_f = 1 - (p_a + p_d)$. The interaction of the thread with the rest of the system is then modelled by the Markov chain depicted in Figure 1.

3 The cautious monitor

We assume the existence of a deterministic Rabin automaton $\mathcal{A} = (Q, 2^{Ap}, \gamma, q_o, Acc)$ for φ. Our monitors keep track of the path π of the product chain $\mathcal{M} \otimes \mathcal{A}$ corresponding to the path of $\mathcal{M}$ executed so far. In order to present the cautious monitor we need some definitions.

Candidate of a path. Given a finite or infinite path $\rho = s_0 s_1 \cdots$ of $\mathcal{M} \otimes \mathcal{A}$, the *support* of ρ is the set $\overline{\rho} = \{s_0, s_1, \ldots\}$. The *graph of* ρ is $G_\rho = (\overline{\rho}, E_\rho)$, where $E_\rho = \{(s_i, s_{i+1}) \mid i = 0, 1, \ldots\}$; i.e., G_ρ has exactly the vertices and edges of $\mathcal{M} \otimes \mathcal{A}$ explored by ρ.

Let π be a finite path of $\mathcal{M} \otimes \mathcal{A}$. If π has a suffix κ such that G_κ is a BSCC of G_π (i.e., the suffix κ of π "looks like" the path π has entered a BSCC), we call $\overline{\kappa}$ the *candidate of* π. Given a path π, we define $K(\pi)$ as follows: If π has a candidate $\overline{\kappa}$, then $K(\pi) := \overline{\kappa}$; otherwise, $K(\pi) := \bot$, meaning that $K(\pi)$ is undefined.

▶ **Example 2.** We have $K(s_0 s_1 s_1) = \{s_1\}$, $K(s_0 s_1 s_1 s_2) = \bot$, and $K(s_0 s_1 s_1 s_2 s_2) = \{s_2\}$ (Figure 1). Also $K(s_0) = K(s_0 s_1) = K(s_0 s_0 s_1) = \bot$, $K(s_0 s_0) = \{s_0\}$, and $K(s_0 s_1 s_0 s_1) = \{s_0, s_1\}$ (Figure 2).

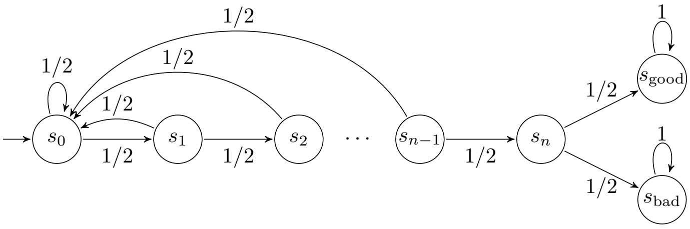


Figure 2 A family of Markov chains.

Good and bad candidates. A candidate K is *good* if $\mathcal{A}$ has a Rabin pair $(E, F) \in Acc$ such that $K \cap (S \times E) = \emptyset$ and $K \cap (S \times F) \neq \emptyset$. Otherwise, K is *bad*. A finite path π of $\mathcal{M} \otimes \mathcal{A}$ is *suspect* if $K(\pi) \neq \bot$ and $K(\pi)$ is a bad candidate. The function $\textsc{Suspect}(\pi)$ returns **true** if π is suspect, and **false** otherwise.

▶ **Proposition 3.**

(a) *Bad runs of $\mathcal{M} \otimes \mathcal{A}$ almost surely have a suspect finite prefix.*

(b) *If the good runs of $\mathcal{M} \otimes \mathcal{A}$ have nonzero probability, then the set of runs without suspect prefixes also has nonzero probability.*

▶ **Example 4** (running). Consider the running example from Figure 1, with $\{s_n\}$ and $\{dead\}$ as good and bad BSCC, respectively. The good paths of the chain are those reaching $\{s_n\}$. They have nonzero probability. The candidate of any bad path is either $\{s_i\}$ for some $0 \leq i \leq n-1$, or the bad BSCC. Since all these candidates are bad, every bad run is suspect. The only good run without suspect prefixes is the one that reaches s_n without ever looping (i.e., all locks are acquired at the first attempt). It has probability p_a^n.

The cautious monitor. The cautious monitor is shown in Algorithm 1. The algorithm samples a run of $\mathcal{M} \otimes \mathcal{A}$ step by step, and restarts whenever the current path π is suspect.

Algorithm 1 $\textsc{CautiousMonitor}$.

```
while true do
    $\pi \leftarrow \lambda$                              ▷ Initialize path with empty path
    repeat
        $\pi \leftarrow \pi\,.\,\mathsf{NextState}(\pi)$                ▷ Extend path
    until $\textsc{Suspect}(\pi)$
```

▶ Remark 5. The cautious monitor only needs to store the *first-appearance record* of π (i.e., the list of states appearing in π, sorted according to their first appearance), since this is enough to compute $\textsc{Suspect}(\pi)$. Even further, it only needs the suffix of length mxsc of the record, where mxsc is the maximal size of the SCCs of the chain. So there exists a universal restart strategy that only needs to store mxsc states. Strategies based on learning the Markov chain, which store all states visited so far, also in previous runs, do not satisfy this property.

To state the correctness of Algorithm 1 formally, we define another Markov chain. Consider the infinite-state Markov chain $\mathcal{C}$ (for cautious) defined as follows. The states of $\mathcal{C}$ are pairs $\langle \pi, r \rangle$, where π is a finite path of $\mathcal{M} \otimes \mathcal{A}$, and $r \geq 0$. Intuitively, r counts the number of restarts so far. The initial probability distribution assigns probability 1 to $\langle \lambda, 0 \rangle$, and 0 to all others. The transition probability matrix $\mathbf{P}_{\mathcal{C}}(\langle \pi, r \rangle, \langle \pi', r' \rangle)$ is defined as follows.

- If π is suspect, then $\mathbf{P}_{\mathcal{C}}(\langle \pi, r\rangle, \langle \pi', r'\rangle) = 1$ if $\pi' = \lambda$ and $r' = r+1$, and 0 otherwise. We call such a transition a *restart*.
- If π is not suspect, then $\mathbf{P}_{\mathcal{C}}(\langle \pi, r\rangle, \langle \pi', r'\rangle) = \mathbf{P}(p, p')$ if $r' = r$, $\pi = \pi'' . p$ and $\pi' = \pi . p'$, and 0 otherwise.

A run of CautiousMonitor corresponds to a run $\rho = \langle \pi_1, r_1\rangle\langle \pi_2, r_2\rangle \cdots$ of $\mathcal{C}$. Let R be the random variable that assigns to ρ the supremum of $r_1, r_2 \cdots$. Further, let S_φ be the set of runs ρ such that $R(\rho) < \infty$ and the suffix of ρ after the last restart satisfies φ. The following theorem states that CautiousMonitor satisfies the properties that were mentioned in the introduction.

▶ **Theorem 6.** *Let φ be an ω-regular property such that $\mathbb{P}[\mathcal{M} \models \varphi] > 0$. Let $\mathcal{C}$ be the Markov chain obtained from $\mathcal{M}$ and φ as above. We have (a) $\mathbb{P}_{\mathcal{C}}[R < \infty] = 1$, and (b) $\mathbb{P}_{\mathcal{C}}[S_\varphi \mid R < \infty] = \mathbb{P}_{\mathcal{C}}[S_\varphi] = 1$.*

Proof. Since $\mathbb{P}[\mathcal{M} \models \varphi] > 0$, the good runs of $\mathcal{M} \otimes \mathcal{A}$ have nonzero probability. By part (b) of Proposition 3, the set of runs of $\mathcal{M} \otimes \mathcal{A}$ without suspect prefixes also has nonzero probability, say p. So, by construction of $\mathcal{C}$, after each restart the probability that $\mathcal{C}$ executes a run of $\mathcal{M} \otimes \mathcal{A}$ without suspect prefixes, and so of executing no more restarts, is at least p. So $\mathbb{P}_{\mathcal{C}}[R = \infty] = 0$. ◀

Performance. Let T be the random variable that assigns to ρ the number of steps till the last restart, or ∞ if the number of restarts is infinite. First of all, we observe that, if we do not make any assumption on the system, $\mathbb{E}(T)$ can grow exponentially in the number of states of the chain. Indeed, consider the family of Markov chains of Figure 2 and the property $\mathbf{F}p$. Assume the only state satisfying p is s_{good}. Then the product of each chain in the family with the DRA for $\mathbf{F}p$ is essentially the same chain, and the good runs are those reaching s_{good}. We show that $\mathbb{E}(T)$ grows at least exponentially for *any* monitor, even for those that have full knowledge of the chain. Indeed, since restarting brings the chain back to s_0, optimal monitors never do a restart when the chain is in any of the states $s_0, \ldots, s_n$; further, they always restart in state s_{bad}, because there is zero probability of reaching s_{good}. So the optimal monitor (i.e., the one with the least expected number of steps) is the one that restarts if, and only when, the chain reaches s_{bad}. Since the chain eventually reaches each of s_{good} and s_{bad} with probability 1/2, the probability that this monitor performs exactly k restarts is $\frac{1}{2^{k+1}}$, and so the average number of restarts is 1. Hence, $\mathbb{E}(T)$ is the expected number of steps needed to reach s_{bad}, under the assumption that it is indeed reached. To reach s_{bad} the chain must execute a run ending with the suffix $s_0 s_1 \ldots s_n$. Since the probability of executing that suffix is $1/2^n$, we get $\mathbb{E}(T) \geq 2^n$. We formulate this result as a proposition.

▶ **Proposition 7.** *Let $\mathcal{M}_n$ be the Markov chain of Figure 2 with n states. Given a monitor $\mathcal{N}$ for the property $\mathbf{F}p$, let $T_{\mathcal{N}}$ be the random variable that assigns to a run of the monitor on $\mathcal{M}_n$ the number of steps till the last restart, or ∞ if the number of restarts is infinite. Then $\mathbb{E}(T_{\mathcal{N}}) \geq 2^n$ for every monitor $\mathcal{N}$.*

This shows that all monitors have bad performance when the time needed to traverse a non-bottom SCC of the chain is very large. So we conduct a parametric analysis in the maximal size mxsc of the SCCs of the chain. This reveals the weak point of CautiousMonitor: $\mathbb{E}(T)$ remains exponential even for families satisfying $\mathsf{mxsc} = 1$. Consider our running example in Figure 1. CautiousMonitor restarts whenever it takes any of the self-loops in a state s_i for $i < n$. Indeed, after that the current path π ends in $s_i s_i$, and so $K(\pi) = \{s_i\}$, which is a bad candidate. So after the last restart the chain must follow the path $s_0 s_1 s_2 \cdots s_n$, which has probability p_a^n, and so $\mathbb{E}(T) \geq p_a^{-n}$. Intuitively, the monitor introduced in the next section is "bolder"; instead of restarting whenever it does not immediately acquire the next lock, it only restarts after trying to acquire a certain number of times, making it "more likely" that the state is *dead*.

4 The bold monitor

We proceed in two steps. In Section 4.1, inspired by [6], we design a bold monitor that knows the minimum probability $\mathsf{p_{min}}$ appearing in $\mathcal{M}$ (more precisely, a lower bound on it). In Section 4.2 we modify this monitor to produce another one that works correctly without any prior knowledge about $\mathcal{M}$, at the price of a performance penalty.

4.1 Chains with known minimal probability

The cautious monitor aborts a run if the current candidate is bad. In contrast, the bold monitor keeps track of the *strength* of the current candidate (defined below), a measure of how confident the monitor can be that the current candidate is a BSCC. In deciding whether a restart should be triggered, the bold monitor also takes into account how many candidates it has already seen since the last restart. Intuitively, the monitor becomes bolder over time, which prevents it from restarting too soon on the family of Figure 1, independently of the length of the chain. The monitor is designed so that it restarts almost all bad runs and only a fixed fraction ε of the good runs. Lemma 11 below shows how to achieve this. We need some additional definitions.

Strength of a candidate and strength of a path. Let π be a finite path of $\mathcal{M} \otimes \mathcal{A}$. The *strength of* $K(\pi)$ in π is undefined if $K(\pi) = \bot$. Otherwise, write $\pi = \pi' \, s \, \kappa$, where $s \in S \times Q$ and π' is the shortest prefix of π such that $K(\pi' s) = K(\pi)$; the strength of $K(\pi)$ is the largest k such that every state of $K(\pi)$ occurs at least k times in $s\,\kappa$, and the last element of $s\,\kappa$ occurs at least $k+1$ times. Intuitively, if the strength is k then every state of the candidate has been been exited at least k times but, for technical reasons, we start counting only after the candidate is discovered. The function $\textsc{Str}(\pi)$ returns the strength of $K(\pi)$ if $K(\pi) \neq \bot$, and 0 otherwise.

▶ **Example 8.** The following table illustrates the definition of strength.

π	$K(\pi)$	π'	s	κ	$\textsc{Str}(\pi)$
p_0p_1	$\bot$	$-$	$-$	$-$	0
$p_0p_1p_1$	$\{p_1\}$	p_0p_1	p_1	ϵ	0
$p_0p_1p_1p_1$	$\{p_1\}$	p_0p_1	p_1	p_1	1
$p_0p_1p_1p_1p_0$	$\{p_0, p_1\}$	$p_0p_1p_1p_1$	p_0	ϵ	0
$p_0p_1p_1p_1p_0p_1$	$\{p_0, p_1\}$	$p_0p_1p_1p_1$	p_0	p_1	0
$p_0p_1p_1p_1p_0p_1p_0$	$\{p_0, p_1\}$	$p_0p_1p_1p_1$	p_0	p_1p_0	1
$p_0p_1p_1p_1p_0p_1p_0p_1p_0$	$\{p_0, p_1\}$	$p_0p_1p_1p_1$	p_0	$p_1p_0p_1p_0$	2

▶ **Example 9** (running). Assume that for the first variable, we need three tries before finally acquiring the lock. Thus, after looping twice in s_0, the strength of the candidate $\{s_0\}$ is 2. If afterwards all variables are acquired on the first try, there are no further candidates until reaching the good BSCC $\{s_n\}$. Then (according to the Markov chain), in state s_n we loop infinitely often and the strength of this candidate goes to infinity in the limit.

Sequence of candidates of a run. Let $\rho = s_0 s_1 \cdots$ be a run of $\mathcal{M} \otimes \mathcal{A}$. Consider the sequence of random variables defined by $K(s_0 \dots s_j)$ for $j \geq 0$, and let $(K_i)_{i \geq 1}$ be the subsequence without undefined elements and with no repetition of consecutive elements. For example, for $\varrho = p_0 p_1 p_1 p_1 p_0 p_1 p_2 p_2 \cdots$, we have $K_1 = \{p_1\}$, $K_2 = \{p_0, p_1\}$, $K_3 = \{p_2\}$, etc. Given a run ρ with a sequence of candidates $K_1, K_2 \dots, K_k$ (observe that this sequence is always finite), we call K_k the final candidate. We define the *strength* of K_i in ρ as the supremum of the strengths of K_i in all prefixes π of ρ such that $K(\pi) = K_i$. For technical convenience, we define $K_\ell := K_k$ for all $\ell > k$ and $K_\infty := K_k$. Observe that ρ satisfies φ iff its final candidate is good. The next lemma follows immediately from the definitions, and the fact that almost surely the runs of a finite-state Markov chain eventually get trapped in a BSCC and visit every state of it infinitely often.

▶ **Lemma 10.** *The final candidate of a run ρ is almost surely a BSCC of $\mathcal{M} \otimes \mathcal{A}$. Moreover, for every k there exists a prefix π_k of ρ such that $K(\pi_k)$ is the final candidate and $\textsc{Str}(\pi_k) \geq k$.*

The bold monitor. The bold monitor for chains with minimal probability $\mathsf{p}_{\mathsf{min}}$ is shown in Algorithm 2. For every ρ and $i \geq 1$, we define two random variables:

- $\textsc{Str}_i(\rho)$ is the strength of $K_i(\rho)$ in ρ;
- $\textsc{Bad}_i(\rho)$ is **true** if $K_i(\rho)$ is a bad candidate, and **false** otherwise.

Let $\alpha_{\mathsf{min}} := -1/\log(1 - \mathsf{p}_{\mathsf{min}})$. The following lemma states that, for every $\alpha \geq \alpha_{\mathsf{min}}$ and $\varepsilon > 0$, the runs that satisfy φ and in which some bad candidate, say K_i, reaches a strength of at least $\alpha(i - \log \varepsilon)$, have probability at most $\varepsilon \mathsf{p}_\varphi$. This leads to the following strategy for the monitor: when the monitor is considering the i-th candidate, abort only if the strength reaches $\alpha(i - \log \varepsilon)$.

▶ **Lemma 11.** *Let φ be an ω-regular property with positive probability p_φ. For every Markov chain $\mathcal{M}$ with minimal probability $\mathsf{p}_{\mathsf{min}}$, for every $\alpha \geq \alpha_{\mathsf{min}}$ and $\varepsilon > 0$:*

$$\mathbb{P}\left[\ \{\rho \mid \rho \models \varphi \wedge \exists i \geq 1 \,.\, \textsc{Bad}_i(\rho) \wedge \textsc{Str}_i(\rho) \geq \alpha(i - \log \varepsilon)\}\ \right] \leq \varepsilon \mathsf{p}_\varphi$$

The monitor is parametric in α and ε. The variable C stores the current candidate, and is used to detect when the candidate changes. The variable i maintains the index of the current candidate, i.e., in every reachable configuration of the algorithm, if $C \neq \bot$ then $C := K_i$.

Algorithm 2 $\textsc{BoldMonitor}_{\alpha,\epsilon}$.

```
while true do
    $\pi \leftarrow \lambda$                                              ▷ Initialize path
    $C \leftarrow \bot$, $i \leftarrow 0$                  ▷ Initialize candidate and candidate counter
    repeat
        $\pi \leftarrow \pi \,.\, \mathsf{NextState}(\pi)$                        ▷ Extend path
        if $\bot \neq K(\pi) \neq C$ then
            $C \leftarrow K(\pi)$; $i \leftarrow i + 1$          ▷ Update candidate and candidate counter
    until $\textsc{Suspect}(\pi)$ and $\textsc{Str}(\pi) \geq \alpha(i - \log \varepsilon)$
```

The infinite-state Markov chain $\mathcal{B}$ of the bold monitor is defined as the chain $\mathcal{C}$ for the cautious monitor; we just replace the condition that π is suspect (and thus has strength at least 1) by the condition that $K(\pi)$ is bad and has strength $\geq \alpha(i - \log \varepsilon)$. The random variable R and the event S_φ are also defined as for $\textsc{CautiousMonitor}$.

▶ **Theorem 12.** *Let $\mathcal{M}$ be a finite-state Markov chain with minimum probability $\mathsf{p}_{\mathsf{min}}$, and let φ be an ω-regular property with probability $\mathsf{p}_\varphi > 0$ in $\mathcal{M}$. Let $\mathcal{B}$ be the Markov chain, defined as above, corresponding to the execution of* BoldMonitor$_{\alpha,\epsilon}$ *on $\mathcal{M} \otimes \mathcal{A}$, where $\alpha \geq \alpha_{\mathsf{min}}$ and $\varepsilon > 0$. We have:*

(a) *The random variable R is geometrically distributed, with parameter (success probability) at least $\mathsf{p}_\varphi(1-\varepsilon)$. Hence, we have $\mathbb{P}_\mathcal{B}[R < \infty] = 1$ and $\mathbb{E}_\mathcal{B}(R) \leq 1/\mathsf{p}_\varphi(1-\varepsilon)$.*

(b) $\mathbb{P}_\mathcal{B}[S_\varphi \mid R < \infty] = \mathbb{P}_\mathcal{B}[S_\varphi] = 1$.

Proof. (a) By Lemma 10, almost all bad runs are restarted. By Lemma 11, runs, conditioned under being good, are restarted with probability at most ε. It follows that the probability that a run is good and not restarted is at least $\mathsf{p}_\varphi(1-\varepsilon)$. (b) In runs satisfying $R < \infty$, the suffix after the last restart almost surely reaches a BSCC of $\mathcal{M} \otimes \mathcal{A}$ and visits all its states infinitely often, increasing the strength of the last candidate beyond any bound. So runs satisfying $R < \infty$ belong to S_φ with probability 1. ◀

In particular, after each restart, the probability that no further restart occurs is at least $\mathsf{p}_\varphi(1-\varepsilon)$. The theoretical optimum would be p_φ, as bad runs need to be restarted almost surely.

Performance. Recall that T is the random variable that assigns to a run the number of steps until the last restart. Let T_j be the number of steps between the j-th and $(j+1)$-st restart. Observe that all the T_j are identically distributed. We have $T_j = T_j^\perp + T_j^C$, where $T_j^\perp$ and T_j^C are the number of prefixes π such that $K(\pi) = \perp$ (no current candidate) and $K(\pi) \neq \perp$ (a candidate), respectively. By deriving bounds on $\mathbb{E}(T_j^\perp)$ and $\mathbb{E}(T_j^C)$, we obtain:

▶ **Theorem 13.** *Let $\mathcal{M}$ be a finite-state Markov chain with minimum probability $\mathsf{p}_{\mathsf{min}}$. Let φ be an ω-regular property with probability $\mathsf{p}_\varphi > 0$ in $\mathcal{M}$. Suppose the product $\mathcal{M} \otimes \mathcal{A}$ has n states and maximal SCC size* mxsc. *Let $\alpha \geq \alpha_{\mathsf{min}}$ and $\varepsilon > 0$. Let T be the number of steps taken by* BoldMonitor$_{\alpha,\epsilon}$ *until the last restart (or ∞ if there are infinitely many restarts, or 0 if there is no restart). We have:*

$$\mathbb{E}(T) \leq \frac{1}{\mathsf{p}_\varphi(1-\varepsilon)} \cdot 2n\,\alpha\,(n - \log\varepsilon)\,\mathsf{mxsc}\left(\frac{1}{\mathsf{p}_{\mathsf{min}}}\right)^{\mathsf{mxsc}}. \tag{1}$$

Observe the main difference with CautiousMonitor: Instead of the exponential dependence on n of Proposition 7, we only have an exponential dependence on mxsc. So if mxsc $\ll n$ the bold monitor performs much better than the cautious one.

4.2 General chains

We adapt BoldMonitor so that it works for arbitrary finite-state Markov chains, at the price of a performance penalty. The main idea is very simple: given any non-decreasing sequence $\{\alpha_n\}_{n=1}^\infty$ of natural numbers such that $\alpha_1 = 1$ and $\lim_{n\to\infty} \alpha_n = \infty$, we sample as in BoldMonitor$_{\alpha,\epsilon}$ but, instead of using the same value α for every sample, we use α_j for the j-th sample. See Algorithm 3, where Sample(α) is the body of the while loop of BoldMonitor$_{\alpha,\varepsilon}$ (lines 2–8 of Algorithm 2) for a given value of α. The intuition is that $\alpha_j \geq \alpha_{\mathsf{min}}$ holds from some index j_0 onwards, and so, by the previous analysis, after the j_0-th restart the monitor almost surely only executes a finite number of restarts. More formally, the correctness follows from the following two properties.

Algorithm 3 BoldMonitor$_\varepsilon$ for $\{\alpha_n\}_{n=1}^{\infty}$.

```
$j \leftarrow 0$
while true do
    $j = j + 1$
    Sample($\alpha_j$)
```

- For every $j \geq 1$, if Sample(α_j) does not terminate then it executes a good run almost surely.
  Indeed, if Sample(α_j) does not terminate then it almost surely reaches a BSCC of $\mathcal{M} \otimes \mathcal{A}$ and visits all its states infinitely often. So from some moment on $K(\pi)$ is and remains equal to this BSCC, and Str(π) grows beyond any bound. Since Sample(α_j) does not terminate, the BSCC is good, and it executes a good run.
- If $\alpha_j \geq \alpha_{\mathsf{min}}$ then the probability that Sample(α_j) does not terminate is at least $\varepsilon \mathsf{p}_\varphi$.
  Indeed, by Lemma 11, if $\alpha_j \geq \alpha_{\mathsf{min}}$, the probability is already at least $\varepsilon \mathsf{p}_\varphi$. Increasing α strengthens the exit condition of the until loop. So the probability that the loop terminates is lower, and the probability of non-termination higher.

These two observations immediately lead to the following proposition:

▶ **Proposition 14.** *Let $\mathcal{M}$ be an arbitrary finite-state Markov chain, and let φ be an ω-regular property such that $\mathsf{p}_\varphi > 0$. Let $\mathcal{B}$ be the Markov chain corresponding to the execution of* BoldMonitor$_\varepsilon$ *on $\mathcal{M} \otimes \mathcal{A}$ with sequence $\{\alpha_n\}_{n=1}^{\infty}$. Let $\mathsf{p}_{\mathsf{min}}$ be the minimum probability of the transitions of $\mathcal{M}$ (which is unknown to* BoldMonitor$_\varepsilon$*). We have*

(a) $\mathbb{P}_\mathcal{B}[R < \infty] = 1$.
(b) $\mathbb{P}_\mathcal{B}[S_\varphi | R < \infty] = \mathbb{P}_\mathcal{B}[S_\varphi] = 1$.
(c) $\mathbb{E}(R) \leq j_{min} + 1/\mathsf{p}_\varphi(1-\epsilon)$*, where j_{min} is the smallest index j such that $\alpha_j \geq \alpha_{\mathsf{min}}$.*

Performance. Different choices of the sequence $\{\alpha_n\}_{n=1}^{\infty}$ lead to versions of BoldMonitor$_\varepsilon$ with different performance features. Intuitively, if the sequence grows very fast, then $j_{\min}$ is very small, and the expected number of restarts $\mathbb{E}(R)$ is only marginally larger than the number for the case in which the monitor knows $\mathsf{p}_{\mathsf{min}}$. However, in this case the last $1/\mathsf{p}_\varphi(1-\epsilon)$ aborted runs are performed for very large values α_j, and so they take many steps. If the sequence grows slowly, then the opposite happens; there are more restarts, but aborted runs have shorter length. Let us analyze two extreme cases: $\alpha_j := 2^j$ and $\alpha_j := j$.

Denote by $f(\alpha)$ the probability that a run is restarted, i.e., the probability that a call Sample(α) terminates. Let $g(\alpha)$ further denote the expected number of steps done in Sample(α) of a run that is restarted (taking the number of steps as 0 if the run is not restarted). According to the analysis underlying Theorem 13, for $\alpha \geq \alpha_{\mathsf{min}}$ we have $g(\alpha) \leq c\alpha$ with $c := 2n\,(n - \log\varepsilon)\,\mathsf{mxsc}\,\mathsf{p}_{\mathsf{min}}^{-\mathsf{mxsc}}$. We can write $T = T_1 + T_2 + \cdots$, where $T_j = 0$ when either the j-th run or a previous run is not restarted, and otherwise T_j is the number of steps of the j-th run. For $j \leq j_{\min}$ we obtain $\mathbb{E}(T_j) \leq g(\alpha_{j_{\min}})$ and hence we have:

$$\mathbb{E}(T) \;=\; \sum_{j=0}^{\infty} \mathbb{E}(T_j) \;\leq\; j_{\min} g(\alpha_{j_{\min}}) + \sum_{i=0}^{\infty} f(\alpha_{j_{\min}})^i g(\alpha_{j_{\min}+i}).$$

By Theorem 12(a) we have $f(\alpha_{j_{\min}}) \leq 1 - \mathsf{p}_\varphi(1-\varepsilon)$. It follows that choosing $\alpha_j := 2^j$ does not in general lead to a finite bound on $\mathbb{E}(T)$. Choosing instead $\alpha_j := j$, we get

$$\mathbb{E}(T) \;\leq\; c j_{\min}^2 + \sum_{i=0}^{\infty} (1 - \mathsf{p}_\varphi(1-\varepsilon))^i c(j_{\min} + i) \qquad \leq\; \left(j_{\min}^2 + \frac{j_{\min}}{\mathsf{p}_\varphi(1-\varepsilon)} + \frac{1}{(\mathsf{p}_\varphi(1-\varepsilon))^2} \right) c\,,$$

where $j_{\min}$ can be bounded by $j_{\min} \leq -1/\log(1-\mathsf{p_{min}})+1 \leq 1/\mathsf{p_{min}}$. So with $c = 2n(n-\log\varepsilon)\mathsf{mxsc}\,\mathsf{p}_{\mathsf{min}}^{-\mathsf{mxsc}}$ we arrive at

$$\mathbb{E}(T) \;\leq\; \left(\frac{1}{\mathsf{p}_{\mathsf{min}}^2} + \frac{1}{\mathsf{p_{min}}\mathsf{p}_\varphi(1-\varepsilon)} + \frac{1}{(\mathsf{p}_\varphi(1-\varepsilon))^2}\right) 2n\,(n-\log\varepsilon)\,\mathsf{mxsc}\left(\frac{1}{\mathsf{p_{min}}}\right)^{\mathsf{mxsc}}, \qquad (2)$$

a bound broadly similar to the one from Theorem 13, but with the monitor not needing to know $\mathsf{p_{min}}$.

4.3 Implementing the bold monitor

A straightforward implementation of the bold monitor in which the candidate $K(\pi)$ and its strength are computed anew each time the path is extended is very inefficient. We present a far more efficient algorithm that continuously maintains the current candidate and its strength. Maintaining them until the path has length n takes $O(n\log n)$ time and $O(s_n \log s_n)$ memory, where s_n denotes the number of states visited by π (which can be much smaller than n when states are visited multiple times). So one update takes $O(\log n)$ amortized time.

Let π be a path of $\mathcal{M}\otimes\mathcal{A}$, and let $s \in \pi$. (Observe that s now denotes a state of $\mathcal{M}\otimes\mathcal{A}$, not of $\mathcal{M}$.) We let $G_\pi = (V_\pi, E_\pi)$ denote the subgraph of $\mathcal{M}\otimes\mathcal{A}$ where V_π and E_π are the sets of states and edges visited by π, respectively. Intuitively, G_π is the fragment of $\mathcal{M}\otimes\mathcal{A}$ explored by the path π.

In the following we define some additional notions related to G_π. Afterwards we show how to use these notions to keep track of the current candidate and its strength during the operation of the bold monitor.

- The *discovery index* of a state s, denoted by $d_\pi(s)$, is the number of states that appear in the prefix of π ending with the first occurrence of s. Intuitively, $d_\pi(s) = k$ if s is the k-th state discovered by π. Since different states have different discovery times, and they do not change when the path is extended, we also call $d_\pi(s)$ the *identifier* of s.
- A *root* of G_π is a state $r \in V_\pi$ such that $d_\pi(r) \leq d_\pi(s)$ for every state $s \in \mathsf{SCC}_\pi(r)$, where $\mathsf{SCC}_\pi(r)$ denotes the SCC of G_π containing r. Intuitively, r is the first state of $\mathsf{SCC}_\pi(r)$ visited by π.
- The *root sequence* R_π of π is the sequence of roots of G_π, ordered by ascending discovery index.
- Let $R_\pi = r_1\, r_2 \cdots r_m$ be the root sequence of π. The sequence $S_\pi = S_\pi(r_1)\, S_\pi(r_2) \cdots S_\pi(r_m)$ of sets is defined by $S_\pi(r_i) := \{s \in V_\pi \mid d_\pi(r_i) \leq d_\pi(s) < d_\pi(r_{i+1})\}$ for every $1 \leq i < m$, i.e., $S_\pi(r_i)$ is the set of states discovered after r_i (including r_i) and before r_{i+1} (excluding r_{i+1}); and $S_\pi(r_m) := \{s \in V_\pi \mid d_\pi(r_m) \leq d_\pi(s)\}$.
- $\mathit{Birthday}_\pi$ is defined as $\bot$ if $K(\pi) = \bot$, and as the length of the shortest prefix π' of π such that $K(\pi') = K(\pi)$ otherwise. Intuitively, $\mathit{Birthday}_\pi$ is the time at which the current candidate of π was created.
- For every state s of π, let π_s be the longest prefix of π ending at s. We define $\mathit{Visits}_\pi(s)$ as the pair $(\mathit{Birthday}_{\pi_s}, v)$, where v is 0 if $\mathit{Birthday}_{\pi_s} = \bot$, and v is the number of times π_s has visited s since $\mathit{Birthday}_{\pi_s}$ otherwise. We define a total order on these pairs: $(b, v) \preceq (b', v')$ iff $b > b'$ (where $\bot > n$ for every number n), or $b = b'$ and $v \leq v'$. Observe that, if π has a candidate, then the smallest pair w.r.t. $\preceq$ corresponds to the state that is among the states visited since the creation of the candidate, and has been visited the least number of times.

The following lemma is an immediate consequence of the definitions.

▶ **Lemma 15.** *Let $G_\pi = (V_\pi, E_\pi)$. The SCCs of G_π are the sets of S_π. Let $S_\pi(r_m)$ be the last set of S_π, and let $(b, v) = \min\{\mathit{Visits}_\pi(s) \mid s \in S_\pi(r_m)\}$, where the minimum is w.r.t. $\preceq$. We have $K(\pi) = \bot$ iff $b = \bot$, and if $b \neq \bot$ then $\textsc{Str}(\pi) = v$.*

By the lemma, in order to efficiently implement BoldMonitor it suffices to maintain R_π, S_π, and a mapping $Visits_\pi$ that assigns $Visits_\pi(s)$ to each state s of π. More precisely, assume that the current path π leads to a state s, and now it is extended to $\pi' = \pi \cdot s'$ by traversing a transition $s \to s'$ of $\mathcal{M} \otimes \mathcal{A}$; it suffices to compute $R_{\pi'}$, $S_{\pi'}$ and $Visits_{\pi'}$ from R_π, S_π, and $Visits_{\pi'}$ in $O(\log n)$ amortized time, where n is the length of π. We first show how to update R_π, S_π, and $Visits_\pi$, and then describe data structures to maintain them in $O(\log n)$ amortized time. We consider four cases:

(1) $s' \notin V_\pi$. That is, the monitor discovers s' by traversing $s \to s'$. Then the SCCs of $G_{\pi'}$ are those of G_π, plus a new trivial SCC containing only s', with s' as root. So $R_{\pi'} = R_\pi \cdot s'$ and $S_{\pi'} = S_\pi \cdot \{s'\}$. Since s' has just been discovered, there is no candidate, and so $Visits_{\pi'}(s') = (\bot, 0)$.

(2) $s' \in V_\pi$ and $d_\pi(s) < d_\pi(s')$. That is, the monitor had already discovered s', and it had discovered it after s. Then $G_{\pi'} = (V_\pi, E_\pi \cup \{(s, s')\})$, but the SCCs of G_π and $G_{\pi'}$ coincide, and so $R_{\pi'} = R_\pi$, $S_{\pi'} = S_\pi$, and if $Visits_\pi(s') = (b, v)$, then $Visits_{\pi'}(s') = (b, v + 1)$.

(3) $s' \in V_\pi$ and $d_\pi(s) = d_\pi(s')$, i.e., $s' = s$. Then as before the SCCs of G_π and $G_{\pi'}$ coincide, and so $R_{\pi'} = R_\pi$, $S_{\pi'} = S_\pi$. If $Visits_\pi(s) = (\bot, 0)$ then $Visits_{\pi'}(s) = (n + 1, 1)$ (recall that n is the length of π), and if $Visits_\pi(s) = (b, v)$ for $b \neq \bot$ then $Visits_{\pi'}(s) = (b, v + 1)$.

(4) $s' \in V_\pi$ and $d_\pi(s) > d_\pi(s')$. That is, the monitor discovered s' before s. Let $R_\pi = r_1\, r_2 \cdots r_m$ and let r_i be the root of $\mathsf{SCC}_\pi(s')$. Then $G_{\pi'}$ has a path $r_i \xrightarrow{*} r_m \xrightarrow{*} s \to s' \xrightarrow{*} r_i$. So, if $Visits_\pi(s') = (b, v)$, we get

$$R_{\pi'} = r_1\, r_2 \cdots r_i \quad S_{\pi'} = S_\pi(r_1) \cdots S_\pi(r_{i-1}) \left(\bigcup_{j=i}^{m} S_\pi(r_j)\right) \quad Visits_{\pi'}(s') = (b, v + 1)$$

In order to efficiently update R_π, S_π and $Visits_\pi$ we represent them using the following data structures.

- The number N of different states visited so far.
- A hash map D that assigns to each state s discovered by π its discovery index. When s is visited for the first time, $D(s)$ is set to $N + 1$. Subsequent lookups return $N + 1$.
- A structure R containing the identifiers of the roots of R_π, and supporting the following operations in amortized $O(\log n)$ time: INSERT(r), which inserts the identifier of r in R; EXTRACT-MAX, which returns the largest identifier in R; and FIND(s), which returns the largest identifier of R smaller than or equal to the identifier of s. (This is the identifier of the root of the SCC containing s.) For example, this is achieved by implementing R both as a balanced search tree and a heap.
- For each root r a structure $S(r)$ representing a set of states and a map assigning to each state s the key $Visits_\pi(s)$, and supporting the following operations in amortized $O(\log n)$ time: FIND-MIN, which returns the minimum value of $Visits_\pi(s)$ over the states of $S(r)$; INCREMENT-KEY(s), which increases the value of the second component of $Visits_\pi(s)$ by 1, and MERGE, which returns the union of two given maps. For example, this is achieved by implementing $S(r)$ as a Fibonacci heap.

▶ **Proposition 16.** *R_π, S_π and $Visits_\pi$ can be updated in $O(\log n)$ amortized time using N, D, R, and S as data structures.*

▶ Remark 17. The implementation above needs $O(s_n \log s_n)$ memory, where s_n denotes the number of states visited by π. Applying the same trick as for the cautious monitor (see Remark 5), the memory consumption can be reduced to $O(\mathsf{mxsc} \log s_n)$, where mxsc is the maximal size of the SCCs of the chain.

5 Experimental results

In this section, we illustrate the behaviour of our monitors on concrete models from the standard PRISM Benchmark Suite [15] and compare the obtained theoretical bounds to the experimental values.

For our implementation, we have re-used the code provided in [6], which in turn is based on the PRISM model checker [14]. Note that whenever the obtained candidate is a good BSCC, no restart will ever happen and we can safely terminate the experiment.

Models. Table 1 lists the models and the properties type, in order of increasing satisfaction probability, which corresponds to increasing optimal number of restarts. The Table contains models from the standard PRISM benchmark suite [15]; the gridworld example, also with the corresponding property negated (written as $\overline{\text{gridworld}}$), in order to represent cases with also very low p_φ; and the example of [6, Fig. 4], called “scale” here, with parameter 10.

Table 1 List of models and types of properties considered. We give the satisfaction probability p_φ, size $|S|$ minimum transition probability $\mathsf{p}_{\mathsf{min}}$, number of bottom SCCs, total number of states in them (an upper bound on mxsc), and number of non-bottom SCCs (NSCC). For the model crowds, the “–” denotes the time-out of the PRISM model checker after 2 hours when trying to compute the characteristics.

Model	Property	p_φ	$\lvert S\rvert$	$\mathsf{p}_{\mathsf{min}}$	#BSCC	BSCC-states	#NSCC
$\overline{\text{gridworld}}$	$\neg(\mathbf{GF} \to \mathbf{FG})$	0.09	309 327	0.001	98	238 053	3
nand	**GF**	0.15	7 014 252	0.02	51	51	0
bluetooth	**GF**	0.20	143 291	0.008	3 072	3,072	24
scale$_{10}$	**GF**	0.50	121	0.5	2	100	20
crowds	**FG**	–	10 633 591	0.067	–	–	–
gridworld	$\mathbf{GF} \to \mathbf{FG}$	0.91	309 327	0.001	98	244 902	1
hermann	**FG**	1.00	524 288	1.9e-6	1	38	9

Compared monitors. The comparison on these models is performed for the following monitors. Firstly, we consider the cautious monitor, which corresponds to the fixed candidate strength 1. Moreover, we also consider the straightforward modification, which requires a different (but still constant) strength, here 10, to practically alleviate the issue with frequent premature restarts. We call these monitors Cautious$_1$ and Cautious$_{10}$. Secondly, we consider the bold monitor using knowledge of the minimal probability $\mathsf{p}_{\mathsf{min}}$, once with low precision of $\varepsilon = 0.5$ and once with a higher one of $\varepsilon = 0.1$, called Bold$_{0.5}$ and Bold$_{0.1}$. Finally, we list the optimal expected number of restarts, corresponding to an ideal (omniscient) monitor, which always makes the correct decision. Note that in our experiments sometimes the average number of restarts is slightly lower than the optimum, since when computing the empirical average of 100 runs, it can vary around the expected value to some extent.

Interpretation of results. Table 2 compares the efficiency of the monitors on the models, in terms of (i) the number of restarts until the monitors leaves the system uninterrupted as it generates a satisfying path, and (ii) the number of steps taken until the last restart when the infinite satisfying run starts. Due to the hugely varying characteristics of the models, we also provide detailed comments on the results model by model in Appendix B.

Table 2 Experimental comparison of the monitors, showing the average number of restarts : average total length of all runs until the final restart. The average is taken over 100 runs of the algorithm. We compare cautious monitors with required candidate strengths 1 and 10, bold monitor with precisions 0.5 and 0.1 and the expected number of restarts for the omniscient monitor. Time-outs are set to two hours, but whenever the monitor finishes successfully, it so happens within a single minute for all considered models. For those models where a time-out occurs, we double-checked three times and the time-out always occurred. Then we ran 100 experiments for one minute and report the average number of restarts per minute.

	$\overline{\text{gridw.}}$	nand	bluetoooth	scale_{10}	crowds	gridw.	hermann
Cautious_1	$TO\ \frac{14000}{\text{min}}$	6.1 : 8 510	4.2 : 2 105	2033 : 6 131	0.98 : 96	0.08 : 3 814	$TO\ \frac{1500}{\text{min}}$
Cautious_{10}	$TO\ \frac{3}{\text{min}}$	4.9 : 6 900	4.8 : 2 425	0.97 : 3 670	1.0 : 101	0.09 : 26 361	$TO\ \frac{1400}{\text{min}}$
$\text{Bold}_{0.5}$	$TO\ \frac{0}{\text{min}}$	5.3 : 8 949	4.2 : 4 851	2.4 : 15 323	1.0 : 220	$TO\ \frac{0}{\text{min}}$	0
$\text{Bold}_{0.1}$	$TO\ \frac{0}{\text{min}}$	5.8 : 10 583	3.7 : 4 637	1.3 : 14 528	1.0 : 199	$TO\ \frac{0}{\text{min}}$	0
$(1/\mathsf{p}_\varphi) - 1$	9.9	5.7	4.1	1.0	?	0.1	0

In summary, the following phenomena can be observed:

- In cases with small BSCCs and few NSCCs (non-bottom SCCs), such as nand or bluetooth, all monitors manage to refute bad BSCCs and find the good one.
- Large BSCCs, as in gridworlds, give a hard time to all monitors. Interestingly, premature restarts of Cautious monitors may quickly yield another chance to hit a good BSCC, where others take long in bad BSCCs to get high confidence to restart. In contrast, premature restarts in intermediate candidates make Cautious monitors keep restarting even in the cases where every simulation goes to the good BSCC, as in hermann.
- Many NSCCs make Cautious_1 restart too often. However, if there is a good chance of leaving the NSCC soon, as in scale_{10}, then the simple modification to Cautious_{10} fixes the issue.
- Since crowds has more than ten million states the PRISM model checker times out after two hours, not yielding information beyond the size. All the simulations are quite short here, indicating the frequent pattern of "fat but shallow" systems. None of the monitors experiences any difficulties, indicating essentially optimal number of restarts (around 1), hence the property seems to be satisfied with roughly 50% probability. The model checker could not conclude that. While the large size prevents a thorough numeric analysis, it did not prevent our monitors from determining satisfaction on single runs.

6 Conclusions

We have presented a universal restart strategy for enforcing arbitrary ω-regular properties in arbitrary finite-state Markov chains. The monitors following the strategy restart the chain whenever the current run is suspect of not satisfying the property. The strategies need no information at all about the chain, its probabilities, or its structure. Contrary to the non-probabilistic settings of [17, 3, 16, 8, 9], they work for arbitrary liveness properties. We have given estimates of the number of steps until the last restart and the total number of steps. The design of dedicated strategies that exploit information on these parameters is an interesting topic for future research.

References

1 Pranav Ashok, Jan Kretínský, and Maximilian Weininger. PAC statistical model checking for Markov decision processes and stochastic games. In *CAV (1)*, volume 11561 of *Lecture Notes in Computer Science*, pages 497–519. Springer, 2019.

2 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

3 David A. Basin, Vincent Jugé, Felix Klaedtke, and Eugen Zalinescu. Enforceable security policies revisited. *ACM Trans. Inf. Syst. Secur.*, 16(1):3:1–3:26, 2013.

4 Hugo Bazille, Blaise Genest, Cyrille Jégourel, and Jun Sun. Global PAC bounds for learning discrete time Markov chains. In *CAV (2)*, volume 12225 of *Lecture Notes in Computer Science*, pages 304–326. Springer, 2020.

5 Yingke Chen, Hua Mao, Manfred Jaeger, Thomas Dyhre Nielsen, Kim Guldstrand Larsen, and Brian Nielsen. Learning Markov models for stationary system behaviors. In *NASA Formal Methods*, volume 7226 of *Lecture Notes in Computer Science*, pages 216–230. Springer, 2012.

6 Przemyslaw Daca, Thomas A. Henzinger, Jan Kretínský, and Tatjana Petrov. Faster statistical model checking for unbounded temporal properties. *ACM Trans. Comput. Log.*, 18(2):12:1–12:25, 2017.

7 Tadashi Dohi, Kishor Trivedi, and Alberto Avritzer, editors. *Handbook of Software Aging and Rejuvenation.* World Scientific, 2020.

8 Ylìès Falcone, Laurent Mounier, Jean-Claude Fernandez, and Jean-Luc Richier. Runtime enforcement monitors: composition, synthesis, and enforcement abilities. *Formal Methods Syst. Des.*, 38(3):223–262, 2011.

9 Ylìès Falcone and Srinivas Pinisetty. On the runtime enforcement of timed properties. In *RV*, volume 11757 of *Lecture Notes in Computer Science*, pages 48–69. Springer, 2019.

10 Matteo Gagliolo and Jürgen Schmidhuber. Learning restart strategies. In *IJCAI*, pages 792–797, 2007.

11 Kalpana Gondi, Yogeshkumar Patel, and A. Prasad Sistla. Monitoring the full range of omega-regular properties of stochastic systems. In *VMCAI*, volume 5403 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 2009.

12 Yennun Huang, Chandra M. R. Kintala, Nick Kolettis, and N. Dudley Fulton. Software rejuvenation: Analysis, module and applications. In *FTCS*, pages 381–390. IEEE Computer Society, 1995.

13 Thomas Jansen. On the analysis of dynamic restart strategies for evolutionary algorithms. In *PPSN*, volume 2439 of *Lecture Notes in Computer Science*, pages 33–43. Springer, 2002.

14 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *CAV*, pages 585–591, 2011.

15 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. The PRISM benchmark suite. In *QEST*, pages 203–204. IEEE Computer Society, 2012.

16 Jay Ligatti, Lujo Bauer, and David Walker. Run-time enforcement of nonsafety policies. *ACM Trans. Inf. Syst. Secur.*, 12(3):19:1–19:41, 2009.

17 Fred B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

18 A. Prasad Sistla and Abhigna R. Srinivas. Monitoring temporal properties of stochastic systems. In *VMCAI*, volume 4905 of *Lecture Notes in Computer Science*, pages 294–308. Springer, 2008.

19 A. Prasad Sistla, Milos Zefran, and Yao Feng. Runtime monitoring of stochastic cyber-physical systems with hybrid state. In *RV*, volume 7186 of *Lecture Notes in Computer Science*, pages 276–293. Springer, 2011.

20 Aad P. A. van Moorsel and Katinka Wolter. Analysis of restart mechanisms in software systems. *IEEE Trans. Software Eng.*, 32(8):547–558, 2006.

21 Jingyi Wang, Jun Sun, Qixia Yuan, and Jun Pang. Should we learn probabilistic models for model checking? A new approach and an empirical study. In *FASE*, volume 10202 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 2017.

22 Katinka Wolter. *Stochastic Models for Fault Tolerance - Restart, Rejuvenation and Checkpointing.* Springer, 2010.

A Technical Proofs

A.1 Proof of Proposition 3

(a) By standard properties of Markov chains, bad runs of $\mathcal{M} \otimes \mathcal{A}$ almost surely reach a BSCC of $\mathcal{M} \otimes \mathcal{A}$ and then traverse all edges of that BSCC infinitely often. Therefore, a bad run ρ almost surely has a finite prefix π that has reached a bad BSCC, say B, of $\mathcal{M} \otimes \mathcal{A}$ and has traversed all edges of B at least once. Then $K(\pi) = B$, and so π is suspect.

(b) Suppose the good runs of $\mathcal{M} \otimes \mathcal{A}$ have nonzero probability. We construct a finite path π, starting at s_0, so that $K(\pi')$ is good for all extensions π' of π, and $K(\pi'')$ is good or undefined for all prefixes π'' of π.

Since the good runs of $\mathcal{M} \otimes \mathcal{A}$ have nonzero probability, $\mathcal{M} \otimes \mathcal{A}$ has a good BSCC B. Let π_1' be a simple (i.e., no repeated states) path from s_0 to a state $s_1 \in B \cap (S \times F)$. Extend π_1' by a shortest path back to the set $\overline{\pi_1'}$ (forming a lasso) and denote the resulting path by π_1. Observe that $K(\pi_1) \subseteq B$ is good, and $K(\pi') = \bot$ holds for all proper prefixes π' of π_1. If $K(\pi_1) = B$, then we can choose $\pi := \pi_1$ and π has the required properties. Otherwise, let π_2' be a shortest path extending π_1 such that π_2' leads to a state in $B \setminus K(\pi_1)$. Extend that path by a shortest path back to $K(\pi_1)$ and denote the resulting path by π_2. Then we have $K(\pi_1) \subsetneq K(\pi_2) \subseteq B$, and $K(\pi_2)$ is good, and $K(\pi') \in \{K(\pi_1), \bot\}$ holds for all paths π' that extend π_1 and are proper prefixes of π_2. Repeat this process until a path π is found with $K(\pi) = B$. This path has the required properties.

A.2 Proof of Lemma 11

Let BSCC denote the set of BSCCs of the chain-automaton product and SCC the set of its SCCs.

For a subset K of states of the product, $\mathit{Cand}_k(K)$ denotes the event (random predicate) of K being a candidate with strength at least k on a run of the product. Further, the "weak" version $\mathit{WCand}_k(K)$ denotes the event that K has strength k when counting visits even prior to discovery of K, i.e. each state of K has been visited and exited at least k times on a prefix π of the run with $K(\pi) = K$. Previous work bounds the probability that a non-BSCC can be falsely deemed BSCC based on the high strength it gets.

▶ **Lemma 18** ([6]). *For every set of states $K \notin \mathsf{BSCC}$, and every $s \in K$, $k \in \mathbb{N}$,*

$$\mathbb{P}_s[\mathit{WCand}_k(K)] \leq (1 - \mathsf{p_{min}})^k .$$

Proof. Since K is not a BSCC, there is a state $t \in K$ with a transition to $t' \notin K$. The set of states K becomes a k-candidate of a run starting from s, only if t is visited at least k times by the path and was never followed by t' (indeed, even if t is the last state in the path, by definition of a k-candidate, there are also at least k previous occurrences of t in the path). Further, since the transition from t to t' has probability at least $\mathsf{p_{min}}$, the probability of not taking the transition k times is at most $(1 - \mathsf{p_{min}})^k$. ◀

In contrast to [6], we need to focus on runs where φ is satisfied. For clarity of notation, we let $K \models \varphi$ denote that K is good, and $K \not\models \varphi$ denote that K is bad. In particular, $K_\infty \models \varphi$ denotes the event that the run satisfies φ.

▶ **Lemma 19.** *For every set of states $K \notin \mathsf{BSCC}$, and every $s \in K$, $k \in \mathbb{N}$,*

$$\mathbb{P}_s[\mathit{WCand}_k(K) \mid K_\infty \models \varphi] \leq (1 - \mathsf{p_{min}})^k .$$

Proof. The previous argument applies also in the case where we assume that after this strength is reached the run continues in any concrete way (also satisfying φ) due to the Markovian nature of the product. In the following derivation, $K\langle t \to t'\rangle$ denotes the event that the candidate K is exited through the transition $t \to t'$:

$$\begin{aligned}
&\mathbb{P}_s[\,WCand_k(K) \mid K_\infty \models \varphi]\\
=&\sum_{t\to t'} \mathbb{P}_s[\,WCand_k(K), K\langle t \to t'\rangle \mid K_\infty \models \varphi]\\
=&\sum_{t\to t'} \mathbb{P}_s[\,WCand_k(K), K\langle t \to t'\rangle, K_\infty \models \varphi]/\mathbb{P}_s[K_\infty \models \varphi]\\
=&\sum_{t\to t'} \mathbb{P}_s[\,WCand_k(K), K\langle t \to t'\rangle] \cdot \mathbb{P}_s[K_\infty \models \varphi \mid WCand_k(K), K\langle t \to t'\rangle] \;/\; \mathbb{P}_s[K_\infty \models \varphi]\\
\overset{(1)}{=}&\sum_{t\to t'} \mathbb{P}_s[\,WCand_k(K), K\langle t \to t'\rangle] \cdot \mathbb{P}_s[K_\infty \models \varphi \mid K\langle t \to t'\rangle] \;/\; \mathbb{P}_s[K_\infty \models \varphi]\\
\overset{(2)}{=}&\sum_{t\to t'} \mathbb{P}_s[\,WCand_k(K), K\langle t \to t'\rangle] \cdot \mathbb{P}_{t'}[K_\infty \models \varphi] \;/\; \mathbb{P}_s[K_\infty \models \varphi]\\
\le&\sum_{t\to t' \text{ exiting } K} \frac{\mathbb{P}_s[\text{reach } t] \cdot \mathbb{P}_t[\text{not take } t \to t' \text{in } k \text{ visits of } t] \cdot \mathbf{P}(t,t') \cdot \mathbb{P}_{t'}[K_\infty \models \varphi]}{\mathbb{P}_s[K_\infty \models \varphi]}\\
=&\sum_{t\to t' \text{ exiting } K} \frac{\mathbb{P}_t[\text{not take } t \to t' \text{in } k \text{ visits of } t] \cdot \mathbb{P}_s[\text{reach } t] \cdot \mathbf{P}(t,t') \cdot \mathbb{P}_{t'}[K_\infty \models \varphi]}{\mathbb{P}_s[K_\infty \models \varphi]}\\
\le&\sum_{t\to t' \text{ exiting } K} \frac{(1-\mathsf{p}_{\mathsf{min}})^k \cdot \mathbb{P}_s[\text{reach } t' \text{ as the first state outside } K] \cdot \mathbb{P}_{t'}[K_\infty \models \varphi]}{\mathbb{P}_s[K_\infty \models \varphi]}\\
=&(1-\mathsf{p}_{\mathsf{min}})^k \cdot \mathbb{P}_s[K_\infty \models \varphi] \;/\; \mathbb{P}_s[K_\infty \models \varphi]\\
=&(1-\mathsf{p}_{\mathsf{min}})^k
\end{aligned}$$

where (1) follows by the Markov property and by (almost surely) $K \neq K_\infty$, (2) by the Markov property. ◀

In the next lemma, we lift the results from fixed designated candidates to arbitrary discovered candidates, at the expense of requiring the (strong version of) strength instead of only the weak strength. To that end, let *birthday* b_i be the moment when ith candidate on a run is discovered, i.e., a run is split into $\rho = \pi b_i \rho'$ so that $K_i = K(\pi b_i) \neq K(\pi)$. In other terms, b_i is the moment we start counting the occurences for the strength, whereas the weak strength is already 1 there.

▶ **Lemma 20.** *For every $i, k \in \mathbb{N}$, we have*

$$\mathbb{P}[Cand_k(K_i) \mid K_i \notin \mathsf{BSCC}, K_\infty \models \varphi] \le (1-\mathsf{p}_{\mathsf{min}})^k\,.$$

Proof.

$$\begin{aligned}
&\mathbb{P}[Cand_k(K_i) \mid K_i \notin \mathsf{BSCC}, K_\infty \models \varphi]\\
=&\frac{\mathbb{P}[Cand_k(K_i), K_i \notin \mathsf{BSCC}, K_\infty \models \varphi]}{\mathbb{P}[K_i \notin \mathsf{BSCC}, K_\infty \models \varphi]}\\
=&\frac{1}{\mathbb{P}[K_i \notin \mathsf{BSCC}, K_\infty \models \varphi]} \sum_{\substack{C \in \mathsf{SCC} \setminus \mathsf{BSCC}\\ s \in C}} \mathbb{P}[Cand_k(C), K_i = C, b_i = s, K_\infty \models \varphi]\\
=&\frac{1}{\mathbb{P}[K_i \notin \mathsf{BSCC}, K_\infty \models \varphi]} \sum_{\substack{C \in \mathsf{SCC} \setminus \mathsf{BSCC}\\ s \in C}} \mathbb{P}[K_i = C, b_i = s] \cdot \mathbb{P}_s[\,WCand_k(C), K_\infty \models \varphi]
\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{\mathbb{P}[K_i \notin \mathsf{BSCC}, K_\infty \models \varphi]} \sum_{\substack{C \in \mathsf{SCC} \setminus \mathsf{BSCC} \\ s \in C}} \mathbb{P}[K_i = C, b_i = s] \cdot \mathbb{P}_s[WCand_k(C) \mid K_\infty \models \varphi] \cdot \mathbb{P}_s[K_\infty \models \varphi] \\
&\leq \frac{(1-\mathsf{p}_{\mathsf{min}})^k}{\mathbb{P}[K_i \notin \mathsf{BSCC}, K_\infty \models \varphi]} \sum_{\substack{C \in \mathsf{SCC} \setminus \mathsf{BSCC} \\ s \in C}} \mathbb{P}[K_i = C, b_i = s] \cdot \mathbb{P}_s[K_\infty \models \varphi] && \text{(by Lemma 19)} \\
&\leq \frac{(1-\mathsf{p}_{\mathsf{min}})^k}{\mathbb{P}[K_i \notin \mathsf{BSCC}, K_\infty \models \varphi]} \sum_{\substack{C \in \mathsf{SCC} \setminus \mathsf{BSCC} \\ s \in C}} \mathbb{P}[K_i = C, b_i = s, K_\infty \models \varphi] \\
&= (1-\mathsf{p}_{\mathsf{min}})^k
\end{aligned}
$$

with the last equality due to

$$
K_i \notin \mathsf{BSCC} \cap K_\infty \models \varphi = \biguplus_{\substack{C \in \mathsf{SCC} \setminus \mathsf{BSCC} \\ s \in C}} K_i = C, b_i = s, K_\infty \models \varphi \qquad ◀
$$

The set $\mathcal{E}rr$ of the next lemma is actually exactly the set considered in Lemma 11 but in a more convenient notation for the computation.

▶ **Lemma 21.** *For $(k_i)_{i=1}^\infty \in \mathbb{N}^\mathbb{N}$, let $\mathcal{E}rr$ be the set of runs such that for some $i \in \mathbb{N}$, we have $Cand_{k_i}(K_i)$ despite $K_i \not\models \varphi$ and $K_\infty \models \varphi$. Then*

$$
\mathbb{P}[\mathcal{E}rr] \leq p_\varphi \sum_{i=1}^\infty (1-\mathsf{p}_{\mathsf{min}})^{k_i} .
$$

Proof.

$$
\begin{aligned}
\mathbb{P}[\mathcal{E}rr] &= \mathbb{P}\left[\bigcup_{i=1}^\infty \Big(Cand_{k_i}(K_i) \cap K_i \not\models \varphi \cap K_\infty \models \varphi\Big)\right] \\
&\leq \mathbb{P}\left[\bigcup_{i=1}^\infty \Big(Cand_{k_i}(K_i) \cap K_i \notin \mathsf{BSCC} \cap K_\infty \models \varphi\Big)\right] \\
&\leq \sum_{i=1}^\infty \mathbb{P}[Cand_{k_i}(K_i) \cap K_i \notin \mathsf{BSCC} \cap K_\infty \models \varphi] && \text{(by the union bound)} \\
&= \sum_{i=1}^\infty \mathbb{P}[Cand_{k_i}(K_i) \mid K_i \notin \mathsf{BSCC} \cap K_\infty \models \varphi] \cdot \mathbb{P}[K_i \notin \mathsf{BSCC} \mid K_\infty \models \varphi] \cdot \mathbb{P}[K_\infty \models \varphi] \\
&\leq \sum_{i=1}^\infty \mathbb{P}[Cand_{k_i}(K_i) \mid K_i \notin \mathsf{BSCC} \cap K_\infty \models \varphi] \cdot 1 \cdot p_\varphi \\
&\leq p_\varphi \sum_{i=1}^\infty (1-\mathsf{p}_{\mathsf{min}})^{k_i} . && \text{(by Lemma 20)}
\end{aligned}
$$

◀

Proof of Lemma 11. Lemma 11 claims that

$$
\mathbb{P}[\mathcal{E}rr] \leq \varepsilon p_\varphi
$$

where k_i is the smallest natural number with $k_i \geq (i - \log \varepsilon) \cdot \frac{-1}{\log(1-\mathsf{p}_{\mathsf{min}})}$. Directly from the previous lemma, by plugging in these k_i, we obtain

$$
\mathbb{P}[\mathcal{E}rr] \leq p_\varphi \sum_{i=1}^\infty (1-\mathsf{p}_{\mathsf{min}})^{k_i} \leq p_\varphi \sum_{i=1}^\infty 2^{-i} 2^{\log \varepsilon} = p_\varphi \varepsilon . \qquad ◀
$$

A.3 Proof of Theorem 13

Let $I_{ji,k}$ be the number of steps between the j-th and $(j+1)$th reset such that the current candidate is K_i, and its strength is k. Observe that for a Markov chain with n states we have $I_{ji,k} = 0$ if $i > n$ or $j > \alpha(i - \log \varepsilon)$. Indeed, if the Markov chain has n states, then along the run there are at most n candidates; moreover, the strength of the K_i stays strictly below $\alpha(i - \log \varepsilon)$, because otherwise the run is aborted. So we have

$$T = \sum_{j=1}^{\infty} T_j = \sum_{j=1}^{\infty} T_j^{\perp} + \sum_{j=1}^{\infty} T_j^{C} = \sum_{j=1}^{\infty} T_j^{\perp} + \sum_{j=1}^{\infty} \sum_{i=1}^{n} \sum_{k=1}^{\alpha(i-\log\varepsilon)} I_{ji,k} \tag{3}$$

and so, by linearity of expectations,

$$\mathbb{E}(T) = \mathbb{E}\left(\sum_{j=1}^{\infty}\left(T_j^{\perp} + \sum_{i=1}^{n}\sum_{k=1}^{\alpha(i-\log\varepsilon)} I_{ji,k}\right)\right) = \mathbb{E}\left(\sum_{j=1}^{\infty} T_j^{\perp}\right) + \sum_{i=1}^{n}\sum_{k=1}^{\alpha(i-\log\varepsilon)}\sum_{j=1}^{\infty} \mathbb{E}\left(I_{ji,k}\right) \tag{4}$$

Let us bound the first summand. Since $K(\pi) = \perp$ only holds when the last state of π is visited for the first time, we have $T_j^{\perp} \leq n$. Moreover, $T_j^{\perp} = 0$ for every $j \geq R$, the number of restarts. So we get

$$\mathbb{E}\left(\sum_{j=1}^{\infty} T_j^{\perp}\right) \leq \mathbb{E}(n \cdot R) = n \cdot \mathbb{E}(R) \tag{5}$$

Consider now the variables $I_{ji,k}$. If $j \geq R$ then $I_{ji,k} = 0$ by definition, since there is no $(j+1)$-th restart. Moreover, under the condition $j < R$ the variables $I_{ji,k}$ and $I_{(j+1)i,k}$ have the same expectation, because they refer to different runs. By Theorem 12(a) R is geometrically distributed with parameter at least $\mathsf{p}_{\varphi}(1-\varepsilon)$, and so we get

$$\mathbb{E}(I_{(j+1)i,k}) \leq \mathbb{E}(I_{ji,k}) \cdot (1 - \mathsf{p}_{\varphi}(1-\varepsilon)) \tag{6}$$

Plugging (4) and (5) into (3), and taking into account that $\mathbb{E}(R) \leq 1/\mathsf{p}_{\varphi}(1-\varepsilon)$, we obtain

$$\begin{aligned}
\mathbb{E}(T) &\leq \mathbb{E}(n \cdot R) + \sum_{i=1}^{n}\sum_{k=1}^{\alpha(i-\log\varepsilon)}\left(\mathbb{E}(I_{0i,k})\sum_{j=0}^{\infty}(1-\mathsf{p}_{\varphi}(1-\varepsilon))^j\right) \\
&= n \cdot \mathbb{E}(R) + \sum_{i=1}^{n}\sum_{k=1}^{\alpha(i-\log\varepsilon)} \frac{\mathbb{E}(I_{0i,k})}{\mathsf{p}_{\varphi}(1-\varepsilon)} \\
&\leq \frac{1}{\mathsf{p}_{\varphi}(1-\varepsilon)}\left(n + \sum_{i=1}^{n}\sum_{k=1}^{\alpha(i-\log\varepsilon)} \mathbb{E}(I_{0i,k})\right)
\end{aligned} \tag{7}$$

If we can find an upper bound $I \geq \mathbb{E}(I_{0i,k})$ for every i, k, then we finally get:

$$\mathbb{E}(T) \leq \frac{1}{\mathsf{p}_{\varphi}(1-\varepsilon)} \cdot n \cdot (1 + \alpha(n - \log\varepsilon) \cdot I) \leq \frac{1}{\mathsf{p}_{\varphi}(1-\varepsilon)} \cdot 2n\alpha(n - \log\varepsilon) \cdot I \tag{8}$$

We now compute a bound $I \geq \mathbb{E}(I_{0i,k})$ valid for arbitrary chains. Recall that $\mathbb{E}(I_{0i,k})$ is the number of steps it takes to increase the strength of the i-th candidate K_i of the 0-th run from k to $k+1$. This is bounded by the number of steps it takes to visit every state of K_i once. Let $\mathsf{mxsc} \in O(n)$ be the maximal size of a SCC. Given any two states s, s' of an SCC, the probability of reaching s' from s after at most mxsc steps is at least $\mathsf{p}_{\mathsf{min}}^{\mathsf{mxsc}}$. So the expected time it takes to visit every state of an SCC at least once is bounded by $\mathsf{mxsc} \cdot \mathsf{p}_{\mathsf{min}}^{-\mathsf{mxsc}}$. So taking $I := \mathsf{mxsc} \cdot \mathsf{p}_{\mathsf{min}}^{-\mathsf{mxsc}}$ we obtain the final result.

A.4 Proof of Proposition 16

When the algorithm explores an edge $s \to s'$ of the Markov chain $\mathcal{M} \otimes \mathcal{A}$, it updates N, D, R, and S as follows. The algorithm first computes $D(s')$, and then proceeds according to the cases (1)-(4):

(1) If s' had not been visited before (i.e., $D(s') = N + 1$), then the algorithm sets $N := N + 1$, inserts $D(s')$ in R, and creates a new Fibonacci heap $S(s')$ containing only the state s' with key $(\bot, 0)$.

(2) If s' had been visited before (i.e., $D(s') \leq N$), and $D(s) < D(s')$, then the algorithm executes FIND(s) to find the root r of the SCC containing s, and updates $S(r)$.

(3) If s' had been visited before (i.e., $D(s') \leq N$), and $D(s) = D(s')$, then the algorithm updates the key of s in $S(s)$.

(4) If s' had been visited before (i.e., $D(s') \leq N$), and $D(s) > D(s')$, then the algorithm executes the following pseudocode, where σ is an auxiliary Fibonacci heap:

```
σ ← ∅
repeat
    r ← EXTRACT-MAX(R)
    σ ← MERGE(σ, S(r))
until D(r) ≤ D(s')
INSERT(r, R)
```

At every moment in time the current candidate is the set $S(r)$, where $r =$ EXTRACT-MAX(R), and its strength can be obtained from FIND-MIN$(S(r))$.

Let us now examine the amortized runtime of the implementation. Let n be the total number of updates, and let $n_1, \ldots, n_4$ be the number of steps executed by the algorithm corresponding to the cases (1)-(4). In cases (1)-(3), the algorithm executes a constant number of heap operations per step, and so it takes $O((n_1 + n_2 + n_3) \log n)$ amortized time for all steps together. This is no longer so for case (4) steps. For example, if the Markov chain is a big elementary circuit $s_0 \to s_1 \to \cdots \to s_{n-1} \to s_0$, then at each step but the last one we insert one state into the heap, and at the last step we extract them all; that is, the last step takes $O(n)$ heap operations. However, observe that each state is inserted in the heap exactly once, when it is discovered, and extracted at most once. So the algorithm executes at most n EXTRACT-MAX and MERGE heap operations for all case (3) steps together, and the amortized time over all of them is $O(n_3 \log n)$. This gives an overall runtime of $O(n \log n)$, and so an amortized time of $O(\log n)$.

B Detailed Experimental Results

We briefly interpret the results for every single model.

gridworld has a very low satisfaction probability and large BSCCs. Hence Bold monitors (and even Cautious$_{10}$) are easily trapped in a large bad BSCC, so it takes very long to realize that with high confidence (also due to small $\mathsf{p}_{\mathsf{min}}$) and restart. In contrast, Cautious$_1$ restarts very often and thus has many chances to get to the good BSCC. Still, the probability of going to the right BSCC and additionally not looping before it has been explored enough to conclude that it satisfies the property is very low, resulting in many restarts and the time-out.

nand also has a low satisfaction probability, but now small BSCCs and no NSCCs (non-bottom SCCs). Hence all monitors manage to refute bad ones and find the good one. Bold monitors spend a bit more time in the bad BSCCs, while Cautious monitors cut earlier. The number of restarts is the same for all (modulo statistical imprecision on 100 runs) since all simply wait for their 1 in 5.7 chance.

bluetooth is similar to nand, but with a few NSCCs. Hence Cautious monitors may occasionally give up on the way, very slightly increasing the number of restarts. But again, Bold monitors need a bit more time in BSCCs to become confident, and thus need more steps.

scale$_{10}$ is designed to have multiple NSCCs on each way, each with 50% self-loop and 50% leaving. Hence Cautious$_1$ can get very easily stuck in one of them, drastically increasing the number of restarts. However, since each NSCC is very probably left within a very few steps, the simple modification to Cautious$_{10}$ fixes the issue, getting essentially the optimum. Bold$_{0.1}$ has no problem with the false restarts, but as it sees many NSCCs, it increases the required strength on the way, resulting in more time to get confidence in the BSCC. Bold$_{0.5}$ has a more relaxed requirements on the strength (starting with less than 10, which is constant for Cautious$_{10}$), making it restart occasionally at the earlier NSCC.

crowds has more than ten million states, making the PRISM model checker time out after two hours, not yielding information beyond the size. All the simulations are quite short, indicating the frequent pattern of “fat but shallow” systems. None of the monitors experiences any difficulties, indicating essentially optimal number of restarts (around 1), hence the property seems to be satisfied with roughly 50%. The model checker could not conclude that. While the large size prevents a thorough numeric analysis, it did not prevent our monitors from determining satisfaction on single runs.

gridworld has a high satisfaction probability and large BSCCs. There is a single NSCC and the good BSCC has 2499 states, hence many simulations just run into the good BSCC. For the one-in-ten chance $(1 - \mathsf{p}_\varphi)$ of reaching a bad BSCC, Cautious monitors just restart, while Bold monitors need to stay there for an impractically large amount of steps to realize they are stuck and want to restart.

hermann satisfies the property almost surely. Bold monitors thus have no problem reaching a good BSCC and exploring the whole of it, which guarantees they will never restart (although the required strength is large, also due to small $\mathsf{p}_{\mathsf{min}}$). In contrast, Cautious monitors keep restarting due to many intermediate candidates, although actually every simulation goes to the good BSCC.

Linear-Time Model Checking Branching Processes

Stefan Kiefer
University of Oxford, UK

Pavel Semukhin
University of Oxford, UK

Cas Widdershoven
University of Oxford, UK

Abstract

(Multi-type) branching processes are a natural and well-studied model for generating random infinite trees. Branching processes feature both nondeterministic and probabilistic branching, generalizing both transition systems and Markov chains (but not generally Markov decision processes). We study the complexity of model checking branching processes against linear-time omega-regular specifications: is it the case almost surely that every branch of a tree randomly generated by the branching process satisfies the omega-regular specification? The main result is that for LTL specifications this problem is in PSPACE, subsuming classical results for transition systems and Markov chains, respectively. The underlying general model-checking algorithm is based on the automata-theoretic approach, using unambiguous Büchi automata.

2012 ACM Subject Classification Theory of computation → Automata over infinite objects; Theory of computation → Verification by model checking

Keywords and phrases model checking, Markov chains, branching processes, automata, computational complexity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.6

Related Version *Full Version*: `https://arxiv.org/abs/2107.01687`

Funding *Stefan Kiefer*: supported by a Royal Society University Research Fellowship.

1 Introduction

Checking whether a (labelled) transition system satisfies a linear-time specification is a staple in verification. The specification is often given as a formula of linear temporal logic (LTL). While early procedures for LTL model checking work directly with the formula [25], the automata-theoretic approach translates LTL formulas into finite automata on infinite words, such as Büchi automata, and analyzes a product of the system and the automaton [37]. This approach can lead to clean and modular model-checking algorithms.

Although LTL captures only a subset of ω-regular languages, model-checking algorithms based on the automata-theoretic approach can be made optimal from the point of view of computational complexity. In particular, model checking finite transition systems against LTL specifications is PSPACE-complete [32], and the algorithm [37] that, loosely speaking, translates (the negation of) the LTL formula into a Büchi automaton and checks the product with the transition system for emptiness can indeed be implemented in PSPACE.

The same approach does not directly work for probabilistic systems modelled as finite Markov chains: intuitively, the nondeterminism in a Büchi automaton causes issues in a stochastic setting where the specification should hold with probability 1, i.e., almost surely but not necessarily surely. A possible remedy is to translate the nondeterministic Büchi automaton further into a deterministic automaton, e.g., a deterministic Rabin automaton (deterministic Büchi automata are less expressive), with which the Markov chain can be naturally instrumented and subsequently analyzed. This determinization step causes a

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 6; pp. 6:1–6:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(second) exponential blowup and does not lead to algorithms that are optimal from a computational-complexity point of view. However, for *Markov decision processes (MDPs)*, which allow for nondeterminism in the probabilistic system, this approach is adequate and leads to an optimal, double-exponential time, model-checking algorithm.

Checking whether a Markov chain satisfies an LTL specification with probability 1 is PSPACE-complete, but membership in PSPACE was proved only in [10, 11], not using the automata-theoretic approach but by a recursive procedure on the formula. This raised the question if there is also an optimal algorithm based on the automata-theoretic approach; see [36] for a survey of the state of the art at the end of the 90s.

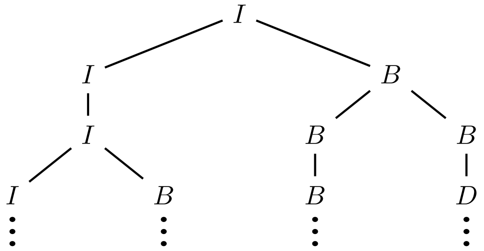
The answer is yes and was first given in [12], using a single-exponential translation from LTL to *separated* Büchi automata. Such automata are special *unambiguous* Büchi automata, which restrict nondeterministic Büchi automata by requiring that every word have at most one accepting run. Another algorithm, using alternating Büchi automata, was proposed in [6], exploiting *reverse determinism*, a property also related to unambiguousness. A polynomial-time (even NC) model-checking algorithm for Markov chains against general unambiguous Büchi automata was given in [2]. These works all imply optimal PSPACE algorithms for LTL model checking of Markov chains via the automata-theoretic approach.

In this paper we exhibit an LTL model checking algorithm that has the following features: (1) it applies to (multi-type) branching processes, a well established model for random trees, generalizing both nondeterministic transition systems and Markov chains; (2) it runs in PSPACE, which is the optimal complexity both for nondeterministic transition systems and Markov chains; and (3) it is based on the automata-theoretic approach (using unambiguous Büchi automata). The fact that there exists an algorithm with the first two features might seem surprising, as one might think that any system model that encompasses both nondeterminism and probability will generalize MDPs, for which LTL model checking is 2EXPTIME-complete [11].

Branching processes (BPs) are a well-studied model in mathematics with applications in numerous fields including biology, physics and natural language processing; see, e.g., [23, 1, 22]. BPs randomly generate infinite trees, and, from a computer-science point of view, they might be the most natural model to do so: (multi-type) BPs can be thought of as a version of stochastic context-free grammars without terminal symbols, randomly generating infinite derivation trees. For example, consider the following BP, taken from [8], with 3 *types* I, B, D:

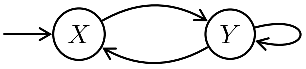
$$\begin{array}{lll} I \xhookrightarrow{0.9} I & B \xhookrightarrow{0.2} D & D \xhookrightarrow{1} D \\ I \xhookrightarrow{0.1} IB & B \xhookrightarrow{0.5} B & \\ & B \xhookrightarrow{0.3} BB & \end{array} \tag{1}$$

This BP might generate a tree with the following prefix:

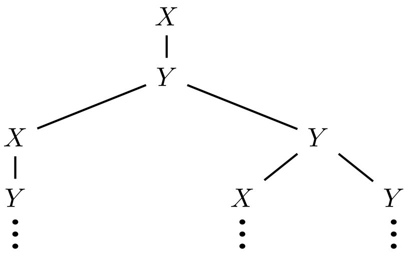


The probability that the BP generates a tree with the shown prefix is the product of the probabilities of the fired transition rules, i.e., (in breadth-first order) $0.1 \cdot 0.9 \cdot 0.3 \cdot 0.1 \cdot 0.5 \cdot 0.2$.

BPs generalize transition systems. Consider the following transition system:

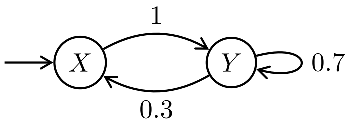


It is equivalent to the BP with $X \overset{1}{\hookrightarrow} Y$ and $Y \overset{1}{\hookrightarrow} XY$, which generates with probability 1 the following unique tree:

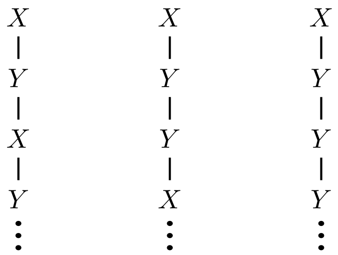


The branches of this unique tree are exactly the executions of the transition system. As a consequence, any LTL formula holds on all executions of the transition system if and only if it holds (with probability 1) on all branches of the generated tree.

BPs also generalize Markov chains. Consider the following Markov chain:



It is equivalent to the BP with $X \overset{1}{\hookrightarrow} Y$ and $Y \overset{0.3}{\hookrightarrow} X$ and $Y \overset{0.7}{\hookrightarrow} Y$, which generates, with probabilities 0.3, $0.7 \cdot 0.3$, $0.7 \cdot 0.7$, respectively, the following prefixes of (degenerated) trees:



Here, each possible "tree" has only a single branch, and the possible "trees" are distributed in the same way as the possible executions of the Markov chain. As a consequence, any LTL formula holds with probability 1 on a random execution of the Markov chain if and only if it holds with probability 1 on the (single) branch of the generated tree.

Hence, both for the transition system and for the Markov chain, the respective model-checking question reduces to the *BP model-checking problem* which asks whether with probability 1 the property holds on *all* branches.

For LTL specifications, we refer to this BP model-checking problem as $\mathbb{P}(\text{LTL}) = 1$. Our main result is that it is in PSPACE, generalizing the corresponding classical results on transition systems and Markov chains. As mentioned, our model-checking algorithm is based on the automata-theoretic approach, in particular on unambiguous Büchi automata. Another important technical ingredient is the algorithmic analysis of certain nonnegative matrices in terms of their spectral radius.

The latter points to the fact that the numbers in the system generally matter, even though we only consider the qualitative problem of comparing the satisfaction probability with 1. For example, for the BP given in (1), one can show that the probability that all branches eventually hit a node of type D is less than 1 (in fact, it is 0). Intuitively, this is because the probability of "branching" via $B \overset{0.3}{\hookrightarrow} BB$ is larger than the probability of "dying" via $B \overset{0.2}{\hookrightarrow} D$. Were the probabilities 0.3 and 0.2 swapped, the probability that all branches eventually hit a node of type D would be 1; cf. [8, Section 1].

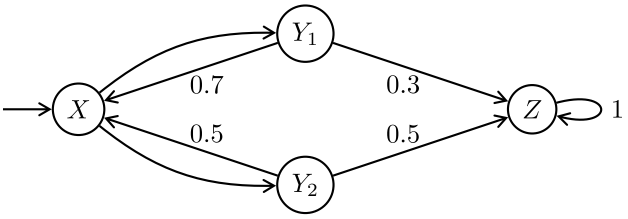
We also consider the problem $\mathbb{P}(\text{LTL} = 0)$, which asks whether the probability that *all* branches satisfy a given LTL formula is 0. Even though it is trivial to negate an LTL formula, this problem is (unlike in Markov chains) not equivalent to the complement of $\mathbb{P}(\text{LTL} = 1)$, because even when the probability is less than 1 that the formula holds on all branches, the probability may still be 0 that the negated formula holds on all branches. We will show that $\mathbb{P}(\text{LTL} = 0)$ is much more computationally complex than $\mathbb{P}(\text{LTL} = 1)$: it is 2EXPTIME-complete.

Besides LTL, we also consider automata-based specifications. Büchi automata are relevant from a verification point of view, as a way of specifying desired or undesired executions of the system. Unambiguous Büchi automata are useful from a technical point of view, in particular, to facilitate our main result on $\mathbb{P}(\text{LTL} = 1)$. See Section 2 and Table 1 for definitions of our problems and a map of our results.

► Remark 1. Readers familiar with MDPs may wonder how the problem $\mathbb{P}(\text{LTL}) = 1$ can have lower computational complexity than the problem whether all schedulers of an MDP satisfy an LTL specification almost surely. Consider the BP

$$X \overset{1}{\hookrightarrow} Y_1Y_2 \qquad Y_1 \overset{0.7}{\hookrightarrow} X \qquad Y_1 \overset{0.3}{\hookrightarrow} Z \qquad Y_2 \overset{0.5}{\hookrightarrow} X \qquad Y_2 \overset{0.5}{\hookrightarrow} Z \qquad Z \overset{1}{\hookrightarrow} Z\,,$$

which might be depicted graphically as follows:



One might view this BP as an MDP where in an X-node the scheduler nondeterministically picks either the Y_1- or the Y_2-successor, and in an Y_i-node, the X- or the Z-successor is chosen randomly. In such an MDP, regardless of the scheduler, a random run reaches with probability 1 a Z-node. However, in the BP above, the probability is positive that *some* branch of a random tree never reaches a Z-node. Although each branch of a random tree could be thought of as being witnessed by at least one scheduler, this is not a contradiction, as there are uncountably many schedulers (over which one cannot take a sum). Hence, if an MDP is interpreted as a BP in the way sketched above, then the requirement that the BP satisfy an LTL formula almost surely on *all* branches is *stronger*, and computationally less complex to check, than the requirement that the MDP satisfy, for *each* scheduler, the formula almost surely.

Related work. We have already discussed related work concerning model checking transition systems and Markov chains.

In addition to the mentioned applications of BPs in various fields, there has also been work on BPs in computer science, especially in the last 10 years. This paper builds on [8], where specifications in terms of deterministic parity *tree* automata are considered. The work [8] implies decidability of the problems considered in this paper and some basic upper complexity bounds. For example, it is not hard to derive from [8] that $\mathbb{P}(\mathrm{LTL} = 1)$ is in 2EXPTIME. Lowering this to PSPACE is the main achievement of this paper.

A related strand of work considers *regular tree languages*; i.e., the specification is not in terms of a word automaton that is run on each branch but in terms of tree automata. Even measurability is not easy to show in this case [20], and fundamental decidability questions around computing the measure have been answered positively only for subclasses of regular tree languages [26, 27].

Fundamental results on the complexity of algorithmically analyzing BPs have been obtained in [18]. Indeed, in Section 3.1 we build on and improve results from [18] on *finiteness* (more often called "extinction" in the literature) of BPs.

Another recent line of work considers extensions of BPs with nondeterminism, focusing on algorithmic questions about properties such as reachability. *Branching MDPs*, which are BPs where a controller chooses actions to influence the evolution of the tree, have been investigated, e.g., in [16, 17]. Even branching *games*, featuring two adversarial controllers, have been studied recently [14].

The work [21] also considers BPs with "internal" nondeterminism (as opposed to the "external" nondeterminism manifested as branching in the generated tree), along with model-checking problems against the logic *GPL*. This expressive, μ-calculus based modal logic had been introduced in [9]. The system model therein, called *reactive probabilistic labeled transition systems (RPLTSs)*, is essentially equivalent to BPs as considered in this paper.

BPs are related to models for probabilistic programs with recursion, such as *Recursive Markov chains*, for which model-checking problems have been studied in detail; see, in particular, [19]. Very loosely speaking, a run of a ("1-exit") Recursive Markov chain can be viewed as a depth-first traversal of a tree generated by a BP. Indeed, for a lower bound in the present paper (Theorem 11) we adapt a proof from [19]. However, most qualitative model-checking problems for Recursive Markov chains are EXPTIME-complete [19], and so many of the BP problems we study turn out to have different computational complexity.

As a key technical tool we use unambiguous Büchi automata, as recently proposed for Markov chains [2]. It is non-trivial to extend their use to random trees, as the branching behaviour of BPs interferes with the spectral-radius based analysis from [2]. One may view as the main technical insight of this paper that the limited nondeterminism in unambiguous automata can be combined with the tree branching of BPs, so that, in a sense, BP model checking reduces to comparing the spectral radius of a certain nonnegative matrix with 1 (Proposition 16).

2 Preliminaries

Let $\mathbb{N}$ and $\mathbb{N}_0$ denote the set of positive and nonnegative integers, respectively. For a finite set Γ, we write Γ^* (resp., Γ^+) for the set of words (resp., nonempty words) over Γ.

Branching processes. A *(multi-type) branching process (BP)* is a tuple $\mathcal{B} = (\Gamma, \hookrightarrow, \mathit{Prob}, X_0)$, where Γ is a finite set of types, $\hookrightarrow \subseteq \Gamma \times \Gamma^+$ is a finite set of transition rules, Prob is a function assigning positive rational probabilities to transition rules so that for every $X \in \Gamma$ we have $\sum_{X \hookrightarrow w} \mathit{Prob}(X \hookrightarrow w) = 1$, and $X_0 \in \Gamma$ is the start type. We write $X \overset{p}{\hookrightarrow} w$ to denote that $\mathit{Prob}(X \hookrightarrow w) = p$. Given a BP $\mathcal{B}$ and a type $X \in \Gamma$ we write $\mathcal{B}[X]$ for the BP obtained from $\mathcal{B}$ by making X the start type. For $X, Y \in \Gamma$ we call Y a *successor* of X if there is a rule $X \hookrightarrow uYv$ for some $u, v \in \Gamma^*$.

A BP *with ε-rules allowed* relaxes the requirement $\hookrightarrow \subseteq \Gamma \times \Gamma^+$ to $\hookrightarrow \subseteq \Gamma \times \Gamma^*$, i.e., there may be rules of the form $X \hookrightarrow \varepsilon$, where ε denotes the empty word. In the following, we disallow ε-rules unless specified otherwise; but the definitions generalize in a natural way.

Fix a BP $\mathcal{B} = (\Gamma, \hookrightarrow, \mathit{Prob}, X_0)$ for the rest of the section.

Trees. Write $[\![\mathcal{B}]\!]$ for the set of trees *generated* by $\mathcal{B}$; i.e., $[\![\mathcal{B}]\!]$ denotes the set of ordered Γ-labelled trees t such that for each $X \in \Gamma$ and each X-labelled node v in t, there is a rule $X \hookrightarrow X_1 \cdots X_k$, denoted by $\mathit{rule}(v)$, such that the k ordered children of v are labelled with $X_1, \ldots, X_k$, respectively. We say a node *has type* $X \in \Gamma$ if the node is labelled with X. A *finite prefix* of a tree $t \in [\![\mathcal{B}]\!]$ is an ordered Γ-labelled *finite* tree obtained from t by designating some nodes as leaves, and removing all their children, grandchildren, etc. Write $(\!|\mathcal{B}|\!)$ for the set of finite prefixes of trees generated by $\mathcal{B}$. For $t \in (\!|\mathcal{B}|\!)$ write $t\!\downarrow \subseteq [\![\mathcal{B}]\!]$ for the ("cylinder") set of trees $t' \in [\![\mathcal{B}]\!]$ such that t is a finite prefix of t'. For $X \in \Gamma$ write $[\![\mathcal{B}]\!]_X \subseteq [\![\mathcal{B}]\!]$ and $(\!|\mathcal{B}|\!)_X \subseteq (\!|\mathcal{B}|\!)$ for the subsets of trees whose root has type X; the trees in $[\![\mathcal{B}]\!]_X$ are called *X-trees.* A *branch* of a tree t is a sequence $v_0 v_1 \cdots$ of nodes in t, where v_0 is the root of t and v_{i+1} is a child of v_i for all $i \in \mathbb{N}_0$. See [8] for equivalent, more formal tree-related definitions.

Probability space. For each $X \in \Gamma$ we define the probability space $([\![\mathcal{B}]\!]_X, \Sigma_X, \mathbb{P}_X)$, where Σ_X is the σ-algebra generated by $\{t\!\downarrow \mid t \in (\!|\mathcal{B}|\!)_X\}$, and $\mathbb{P}_X$ is the probability measure generated by $\mathbb{P}_X(t\!\downarrow) := \prod_v \mathit{Prob}(\mathit{rule}(v))$ for all $t \in (\!|\mathcal{B}|\!)_X$, where the product extends over all non-leaf nodes v in t. This is analogous to the standard definition of the probability space of a Markov chain. We may write $\mathbb{P}_{\mathcal{B}}$ for $\mathbb{P}_{X_0}$, omitting the subscript when $\mathcal{B}$ is understood. We often talk about events (i.e., measurable sets of trees) and their probability in text form. For example, by saying "a $\mathcal{B}$-tree has with positive probability infinitely many nodes of type X" we mean that $\mathbb{P}_{\mathcal{B}}(E) > 0$ where $E \subseteq [\![\mathcal{B}]\!]_{X_0}$ is the set of X_0-trees with infinitely many nodes of type X.

Linear-Time Properties. We are particularly interested in sets of trees *all whose branches* (more precisely, their associated sequences of types) satisfy an ω-regular linear-time property $L \subseteq \Gamma^\omega$. Given $L \subseteq \Gamma^\omega$, we write $\mathbb{P}_{\mathcal{B}}(L)$ for the probability that *all* branches of a $\mathcal{B}$-tree satisfy L. *Linear temporal logic (LTL)* formulas specify linear-time properties; see, e.g., [34] for a definition of LTL. An important example for us are formulas of the form $\mathsf{F}T$, where $T \subseteq \Gamma$, which denotes the linear-time property $\{uXw \mid u \in \Gamma^*,\ X \in T,\ w \in \Gamma^\omega\}$. Accordingly, $\mathbb{P}_{\mathcal{B}}(\mathsf{F}T)$ denotes the probability that all branches of a $\mathcal{B}$-tree have a node whose type is in T (equivalently, the probability that a $\mathcal{B}$-tree has a finite prefix all whose leaves have a type in T).

Automata. We use finite automata on infinite words over Γ, where Γ is the set of types of a BP. We use *deterministic parity automata (DPAs)*, *deterministic Büchi automata (DBAs)*, *nondeterministic Büchi automata (NBAs)*, and *unambiguous Büchi automata (UBAs)*. The definitions are standard; see, e.g., [34]. In the following we fix some terms and notation. Let $\mathcal{A} = (Q, \Gamma, \delta, Q_0, F)$ be an NBA, where Q is a finite set of states, Γ is the alphabet, $\delta \subseteq Q \times \Gamma \times Q$ is the transition relation, $Q_0 \subseteq Q$ is the set of initial states, and $F \subseteq Q$ is the set of accepting states. We write $q \xrightarrow{X} r$ to denote that $(q, X, r) \in \delta$. A finite sequence $q_0 \xrightarrow{X_1} q_1 \xrightarrow{X_2} \cdots \xrightarrow{X_n} q_n$ is called a *path* and can be summarized as $q_0 \xrightarrow{X_1 \cdots X_n}{}^{*} q_n$. An

Table 1 Results and organization of the paper. The complexity classes indicate completeness results, except "in NC", which only means membership in NC.

	$= 1$	$= 0$
$\mathbb{P}$(finite)	in NC	
Section 3.1	Proposition 6	
$\mathbb{P}$(DPA)	in NC	P
Section 3.2	Theorem 8	Theorem 9
$\mathbb{P}$(NBA)	PSPACE	EXPTIME
Section 3.3	Theorem 10	Theorem 11
$\mathbb{P}$(coNBA)	PSPACE	EXPTIME
Section 4	Theorem 14	Theorem 15
$\mathbb{P}$(coUBA)	in NC	
Section 5	Proposition 16	
$\mathbb{P}$(LTL)	PSPACE	2EXPTIME
Section 6	Theorem 18	Theorem 19

infinite sequence $q_0 \xrightarrow{X_1} q_1 \xrightarrow{X_2} \cdots$ is called a *run* of $X_1 X_2 \cdots$. We call the run *accepting* if $q_0 \in Q_0$ and $q_i \in F$ holds for infinitely many q_i. The NBA $\mathcal{A}$ *accepts* (resp., *rejects*) an infinite word $w \in \Gamma^\omega$ if w has (resp., does not have) an accepting run in $\mathcal{A}$. The NBA $\mathcal{A}$ is called an *unambiguous Büchi automaton (UBA)* if every $w \in \Gamma^\omega$ has at most one accepting run. An automaton $\mathcal{A}$ defines ω-regular linear-time properties $\{w \in \Gamma^\omega \mid \mathcal{A} \text{ accepts } w\}$ and $\{w \in \Gamma^\omega \mid \mathcal{A} \text{ rejects } w\}$. In keeping with previous definitions, we write $\mathbb{P}_\mathcal{B}(\mathcal{A} \text{ accepts})$ (resp., $\mathbb{P}_\mathcal{B}(\mathcal{A} \text{ rejects})$) for the probability that *all* branches of a $\mathcal{B}$-tree (more precisely, their associated sequences of types) are accepted (resp., rejected) by $\mathcal{A}$.

Problems. We consider the following computational problems. The problem $\mathbb{P}(\text{finite}) = 1$ asks, given a BP $\mathcal{B}$ with ε-rules allowed, whether the probability that a $\mathcal{B}$-tree is finite is 1. The problem $\mathbb{P}(\text{LTL}) = 1$ asks, given a BP $\mathcal{B}$ and an LTL formula φ, whether $\mathbb{P}_\mathcal{B}(\varphi) = 1$. The problems $\mathbb{P}(\text{DPA}) = 1$ (resp., $\mathbb{P}(\text{NBA}) = 1$) ask, given a BP $\mathcal{B}$ and a DPA (resp., NBA) $\mathcal{A}$, whether $\mathbb{P}_\mathcal{B}(\mathcal{A} \text{ accepts}) = 1$. The problems $\mathbb{P}(\text{coNBA}) = 1$ (resp., $\mathbb{P}(\text{coUBA}) = 1$)[1] ask, given a BP $\mathcal{B}$ and an NBA (resp., UBA) $\mathcal{A}$, whether $\mathbb{P}_\mathcal{B}(\mathcal{A} \text{ rejects}) = 1$. The problems $\mathbb{P}(\text{LTL}) = 0, \mathbb{P}(\text{DPA}) = 0, \ldots$ are defined similarly, where "$=1$" is replaced with "$=0$". See Table 1 for a map of our results in those terms, as well as for an overview of the rest of the paper. As explained in the introduction, the problem $\mathbb{P}(\text{LTL}) = 1$ is of particular interest from a model-checking point of view, and the technically most challenging one.

Complexity Classes. In addition to standard complexity classes between P and 2EXPTIME, we use the class NC, the subclass of P comprising those problems solvable in polylogarithmic time by a parallel random-access machine using polynomially many processors; see, e.g., [28, Chapter 15]. To prove membership in PSPACE in a modular way, we will use the following pattern:

▶ **Lemma 2.** *Let P_1, P_2 be two problems, where P_2 is in NC. Suppose there is a reduction from P_1 to P_2 implemented by a PSPACE transducer, i.e., a Turing machine whose work tape (but not necessarily its output tape) is PSPACE-bounded. Then P_1 is in PSPACE.*

Proof. Note that the output of the transducer is (at most) exponential. Problems in NC can be decided in polylogarithmic space [4, Theorem 4]. Using standard techniques for composing space-bounded transducers (see, e.g., [28, Proposition 8.2]), it follows that P_1 is in PSPACE. ◀

[1] We do not explicitly define or use a notion of "co-Büchi automata" to avoid possible confusion about accepting/rejecting. If one were to do so, one would define a "co-NBA" $\mathcal{A}$ like an NBA $\mathcal{A}$, but the "co-NBA" $\mathcal{A}$ would accept a word $w \in \Gamma^\omega$ if and only if $\mathcal{A}$ viewed as an NBA rejects w. Similarly for "co-UBAs".

Matrices. We use finite sets S to index matrices $M \in \mathbb{R}^{S \times S}$ and vectors $v \in \mathbb{R}^S$. The *graph* of a nonnegative matrix $M \in [0, \infty)^{S \times S}$ is the directed graph (S, E) with $E = \{(s,t) \in S \times S \mid M_{s,t} > 0\}$. The *spectral radius* of a matrix is the largest absolute value of its eigenvalues. The following lemma allows to efficiently compare the spectral radius of a nonnegative matrix with 1.

▶ **Lemma 3.** *Given a nonnegative rational matrix M, one can determine in NC whether $\rho < 1$ or $\rho = 1$ or $\rho > 1$, where ρ denotes the spectral radius of M.*

Proof. Use the algorithm from [13, Proposition 2.2], but not with Gaussian elimination as suggested there, but by solving the systems of linear equations described in [13, Proposition 2.2] in NC. The latter is possible in NC [5, Theorem 5]. ◀

3 Basic Results

In this section we develop the more basic results indicated in Table 1, on finiteness (Section 3.1), deterministic parity automata (Section 3.2), and Büchi automata (Section 3.3), on the one hand rounding off the complexity map in Table 1, and on the other hand building the foundation for more challenging results in the following sections. In particular, Proposition 6 is indirectly used throughout the paper.

3.1 Finiteness

In this section we consider BPs with ε-rules allowed, i.e., rules of the form $X \hookrightarrow \varepsilon$. Such BPs may generate finite trees. We are interested in the *almost-sure finiteness* problem, also denoted as $\mathbb{P}(\text{finite}) = 1$, i.e., the problem whether the probability that a given BP with ε-rules allowed generates a finite tree is equal to 1. In Proposition 6 below we show that this problem is in NC. All upper bounds on the complexity of $\mathbb{P}(\cdot) = 1$ problems in this paper build directly or indirectly on this result.

While the almost-sure finiteness (or "extinction") problem has often been studied and is known to be in (strongly) polynomial time [18, 13], its membership in NC is, to the best of the authors' knowledge, new. For instance, since linear programming is P-complete, one cannot use linear programming (as in [18]) to show membership in NC. Nor can one directly use the strongly polynomial-time algorithm of [13], as it computes, in a sub-procedure, the set of types X for which there *exists* a finite X-tree. But the latter problem is P-complete.

For the rest of the section, fix a BP $\mathcal{B} = (\Gamma, \hookrightarrow, \mathit{Prob}, X_0)$ with ε-rules allowed. Define a directed graph $G = (\Gamma, E)$ (i.e., the types of $\mathcal{B}$ are the vertices of G) with an edge $(X, Y) \in E$ if and only if Y is a successor of X (i.e., there is a rule $X \hookrightarrow uYv$ for some $u, v \in \Gamma^*$). Given a strongly connected component (SCC) $S \subseteq \Gamma$ of G and $X \in S$, define a BP $\mathcal{B}[S, X] = (S, \hookrightarrow_S, \mathit{Prob}_S, X)$ obtained from $\mathcal{B}$ by restricting the types to S and deleting on all right-hand sides of the rules those types not in S. The following lemma is straightforward:

▶ **Lemma 4.** *A $\mathcal{B}$-tree is infinite with positive probability if and only if there exist an SCC $S \subseteq \Gamma$ of G and $X \in S$ such that X is reachable from X_0 in G and a $\mathcal{B}[S, X]$-tree is infinite with positive probability.*

Let $M \in \mathbb{Q}^{\Gamma \times \Gamma}$ be the nonnegative $\Gamma \times \Gamma$-matrix with $M_{X,Y} = \sum_{X \overset{p}{\hookrightarrow} w} p|w|_Y$, where $|w|_Y \in \mathbb{N}_0$ is the number of occurrences of Y in w. That is, $M_{X,Y}$ is the expected number of direct Y-successors of the root of a $\mathcal{B}[X]$-tree. By induction, M^i, the ith power of M, is such that $(M^i)_{X,Y}$ is the expected number of Y-nodes that are exactly i levels under the root of a $\mathcal{B}[X]$-tree. The graph of M is exactly the previously defined graph G.

Let $S \subseteq \Gamma$ be an SCC of G. Denote by $M_S \in \mathbb{Q}^{S \times S}$ the (square) principal submatrix obtained from M by restricting it to the rows and columns indexed by elements of S. Let ρ_S denote the spectral radius of M_S. Call S *supercritical* if $\rho_S > 1$. Call S *linear* if for all rules $X \hookrightarrow w$ with $X \in S$ there is exactly one occurrence in w of a type in S. Observe that if S is linear then M_S is *stochastic*, i.e., $M_S \vec{1} = \vec{1}$ where $\vec{1}$ is the all-1 vector, i.e., the element of $\{1\}^S$. In that case, by the Perron-Frobenius theorem [3, Theorem 2.1.4 (b)], we have $\rho_S = 1$ and, thus, S is not supercritical.

The following characterization can be proved using [13, Section 3] (which builds on [18, Section 8.1]):

▶ **Lemma 5.** *A $\mathcal{B}$-tree is infinite with positive probability if and only if there exist an SCC $S \subseteq \Gamma$ of G and $X \in S$ such that X is reachable from X_0 in G and S is supercritical or linear.*

It follows:

▶ **Proposition 6.** *The problem $\mathbb{P}(\mathit{finite}) = 1$ is in NC.*

3.2 Deterministic Parity Automata

In this section we consider deterministic parity automata (DPAs) on words. In [8, Section 3] it was shown that the problem $\mathbb{P}(\mathrm{DPA}) = 1$ can be decided in polynomial time. We improve this to membership in NC.

By the following lemma we can check in NC whether a $\mathcal{B}$-tree almost surely has a finite prefix all whose leaves have types in a given set T. The proof is by reduction to almost-sure finiteness.

▶ **Lemma 7.** *Given a BP $\mathcal{B} = (\Gamma, \hookrightarrow, \mathit{Prob}, X_0)$ and a set of types $T \subseteq \Gamma$, the problem whether $\mathbb{P}_{X_0}(\mathsf{F}T) = 1$ is in NC.*

By combining Lemma 7 with results from [8] we obtain:

▶ **Theorem 8.** *The problem $\mathbb{P}(\mathrm{DPA}) = 1$ is in NC.*

The hardness result in the following theorem highlights the different complexities of $\mathbb{P}(\cdot) = 0$ and $\mathbb{P}(\cdot) = 1$ problems in this paper.

▶ **Theorem 9.** *The problem $\mathbb{P}(\mathrm{DPA}) = 0$ is P-complete. It is P-hard even for deterministic Büchi automata with two states, the accepting state being a sink.*

3.3 Büchi Automata

▶ **Theorem 10.** *The problem $\mathbb{P}(\mathrm{NBA}) = 1$ is PSPACE-complete.*

Proof. PSPACE-hardness is immediate in two different ways. It follows from the PSPACE-hardness of model checking Markov chains against NBAs [35]. It also follows from the PSPACE-hardness of model checking transition systems against NBAs. (The latter follows easily from the PSPACE-hardness of NBA universality [32].) Both model-checking problems are special cases of $\mathbb{P}(\mathrm{NBA}) = 1$.

Towards membership in PSPACE, we use a translation from NBA to DPA [29]. This translation causes an exponential blow-up, but an inspection of the construction [29, Section 3.2] reveals that it can be computed by a PSPACE transducer. By Theorem 8 the problem $\mathbb{P}(\mathrm{DPA}) = 1$ is in NC. By Lemma 2 it follows that $\mathbb{P}(\mathrm{NBA}) = 1$ is in PSPACE. ◀

▶ **Theorem 11.** *The problem* $\mathbb{P}(\mathrm{NBA}) = 0$ *is EXPTIME-complete. It is EXPTIME-hard even for NBAs whose only accepting state is a sink.*

Proof. Towards membership in EXPTIME, an NBA can be translated, in exponential time, to a DPA of exponential size; see, e.g., [29]. Since $\mathbb{P}(\mathrm{DPA}) = 0$ is in P by Theorem 9, it follows that $\mathbb{P}(\mathrm{NBA}) = 0$ is in EXPTIME.

Concerning EXPTIME-hardness, we adapt the proof (in the online appendix) of [19, Theorem 17] on model checking *recursive Markov chains* against NBAs. The details are in [24]. ◀

4 Co-Büchi Automata

In this section we consider the problem $\mathbb{P}(\mathrm{coNBA}) = 1$, which asks, given a BP $\mathcal{B}$ and a Büchi automaton $\mathcal{A}$, whether $\mathcal{B}$ almost surely generates a tree whose branches are all rejected by $\mathcal{A}$; i.e., whether $\mathbb{P}_{\mathcal{B}}(\mathcal{A} \text{ rejects}) = 1$. Dually, one might ask whether the probability is positive that a $\mathcal{B}$-tree has a branch accepted by $\mathcal{A}$. Intuitively, we view the Büchi automaton $\mathcal{A}$ as specifying "bad" branches, and we would like the tree almost surely not to have any bad branches.

This problem is in PSPACE, which can be shown via a translation to DPAs, as in Theorem 10. However, with a view on the following sections, in particular on LTL specifications, we pursue a different approach to the problem $\mathbb{P}(\mathrm{coNBA}) = 1$. In this section we lay the groundwork for arbitrary Büchi automata $\mathcal{A}$. By building on these results, we will show in the next section that if $\mathcal{A}$ is *unambiguous* then the problem is in NC, which will allow us to derive our headline result, namely that $\mathbb{P}(\mathrm{LTL}) = 1$ is in PSPACE.

Let $\mathcal{B} = (\Gamma, \hookrightarrow, \mathit{Prob}, X_0)$ be a BP and $\mathcal{A} = (Q, \Gamma, \delta, Q_0, F)$ a (not necessarily unambiguous) Büchi automaton.

Define a Büchi automaton, $\mathcal{A} \times \mathcal{B}$, by $\mathcal{A} \times \mathcal{B} := (Q \times \Gamma, \Gamma, \delta_{\mathcal{A}\times\mathcal{B}}, Q_0 \times \{X_0\}, F \times \Gamma)$, where

$$\delta_{\mathcal{A}\times\mathcal{B}}((q_1, X_1), X_2) = \begin{cases} \delta(q_1, X_1) \times \{X_2\} & \text{if } X_2 \text{ is a successor of } X_1 \\ \emptyset & \text{otherwise.} \end{cases}$$

The remainder of the section is organized as follows. In Section 4.1 we show that the problem $\mathbb{P}(\mathrm{coNBA}) = 1$ reduces to the analysis of certain SCCs within $\mathcal{A} \times \mathcal{B}$. In Section 4.2 we introduce a key lemma, Lemma 13, which allows us to "forget" about the distinction between accepting and non-accepting states: the lemma reduces $\mathbb{P}(\mathrm{coNBA}) = 1$ to a pure reachability problem in an exponential-sized BP, $\mathcal{B}_{det}$. This leads us to prove PSPACE-completeness of $\mathbb{P}(\mathrm{coNBA}) = 1$, but more importantly, Lemma 13 plays a key role in the rest of the paper. We prove it in [24].

4.1 The Automaton $\mathcal{A}[f, X_f]$

For any $(f, X_f) \in F \times \Gamma$ on a cycle of the transition graph of $\mathcal{A} \times \mathcal{B}$, define the Büchi automaton

$$\mathcal{A}[f, X_f] \;:=\; (\{\bar{q}_0\} \cup Q[f, X_f], \Gamma, \delta[f, X_f], \{\bar{q}_0\}, \{(f, X_f)\})$$

as the Büchi automaton obtained from $\mathcal{A} \times \mathcal{B}$ by

1. making (f, X_f) the only accepting state,
2. restricting the set of states, $Q[f, X_f] \subseteq Q \times \Gamma$, to those (q, X) that, in the transition graph of $\mathcal{A} \times \mathcal{B}$, are reachable from (f, X_f) and can reach (f, X_f), i.e., those (q, X) in the SCC containing (f, X_f),

3. restricting the transition function $\delta[f, X_f]$ accordingly, i.e.,

$$\delta[f, X_f]((q, X), Y) \ := \ \delta_{\mathcal{A}\times\mathcal{B}}((q, X), Y) \cap Q[f, X_f]\,,$$

4. making $\bar{q}_0$ the only initial state, and
5. setting $\delta[f, X_f](\bar{q}_0, X_f) := \{(f, X_f)\}$ and $\delta[f, X_f](\bar{q}_0, X) := \emptyset$ for all $X \in \Gamma \setminus \{X_f\}$.

The following lemma follows from the pigeonhole principle and basic probability arguments:

▶ **Lemma 12.** *The probability that some branch of a $\mathcal{B}$-tree is accepted by $\mathcal{A}$ is positive if and only if there are $q_0 \in Q_0$ and $f \in F$ and $X_f \in \Gamma$ such that (f, X_f) is reachable from (q_0, X_0) in the transition graph of $\mathcal{A} \times \mathcal{B}$ and the probability that some branch of a $\mathcal{B}[X_f]$-tree is accepted by $\mathcal{A}[f, X_f]$ is positive.*

For the rest of the section let $(f, X_f) \in F \times \Gamma$ be on a cycle of the transition graph of $\mathcal{A} \times \mathcal{B}$.

4.2 The Determinization $\mathcal{A}_{det}$ and the BP $\mathcal{B}_{det}$

Let

$$\mathcal{A}_{det} := (2^{\{\bar{q}_0\}\cup Q[f,X_f]}, \Gamma, \delta_{det}, \{\bar{q}_0\}, 2^{\{\bar{q}_0\}\cup Q[f,X_f]} \setminus \{\emptyset\})$$

be the determinization of $\mathcal{A}[f, X_f]$ obtained by the standard subset construction. Which states are accepting will not actually be relevant. Note that every state reachable via a nonempty path from $\{\bar{q}_0\}$ is of the form $P \times \{X\}$ with $P \subseteq Q$ and $X \in \Gamma$.

Define a BP $\mathcal{B}_{det}$ based on $\mathcal{A}_{det}$ as

$$\mathcal{B}_{det} \ := \ (\Gamma', \hookrightarrow', \mathit{Prob}', \{(f, X_f)\})\,,$$

where the set of types $\Gamma' \subseteq 2^{Q[f,X_f]}$ is the set of those states in $\mathcal{A}_{det}$ that are reachable (in $\mathcal{A}_{det}$) from $\{\bar{q}_0\}$ via a nonempty path (recall that they are of the form $P \times \{X\}$ with $P \subseteq Q$ and $X \in \Gamma$), and

$$X' \overset{p}{\hookrightarrow}{}' \ \delta_{det}(X', X_1) \cdots \delta_{det}(X', X_k)$$

for all $X' = P \times \{X\} \in \Gamma'$ with $P \neq \emptyset$ and all $X \overset{p}{\hookrightarrow} X_1 \cdots X_k$, and $\emptyset \overset{1}{\hookrightarrow}{}' \emptyset$. Here is the key lemma of this section:

▶ **Lemma 13.** *The following statements are equivalent:*
(i) *The probability that some branch of a $\mathcal{B}[X_f]$-tree is accepted by $\mathcal{A}[f, X_f]$ is positive.*
(ii) *The probability that some branch of a $\mathcal{B}_{det}$-tree does not have any nodes of type $\emptyset$ is positive.*

We prove Lemma 13 in [24]. It will be used in the proof of Theorem 14 below; but more importantly, Lemma 13 is the foundation of Section 5.

Given that Lemma 13 reflects the key insight of this section, let us comment further. Considering that condition (ii) does not mention a notion of acceptance, one might have two concerns at this point:

(a) Condition (ii) does not obviously imply that with positive probability there is even a branch with infinitely many nodes of types containing (f, X_f).
(b) Even if with positive probability there is such a branch, it is not obvious that such branches would necessarily correspond to branches of $\mathcal{B}[X_f]$ that are accepted by $\mathcal{A}[f, X_f]$.

Even for the special case of Markov chains (i.e., every tree has only a single branch), Lemma 13 is not at all obvious, and both concerns (a) and (b) apply. Indeed, for Markov chains, Courcoubetis and Yannakakis prove a statement related to Lemma 13, namely [11, Proposition 4.1.4], with a proof related to ours and dealing explicitly with concern (b) above. For the special case of transition systems (i.e., the BP generates exactly one tree), Lemma 13 is simple though: consider the branch that follows a cycle around (f, X_f). For the general case, we need a result on BPs from [8], dealing with concern (a) above. The high-level principle behind the proof of Lemma 13 is often used in the analysis of Markov chains: if it is possible, infinitely often, to reach a state with a probability bounded away from 0, then this state is almost surely reached infinitely often. See [24] for a full proof of Lemma 13.

We can now derive a PSPACE procedure for the problem $\mathbb{P}(\text{coNBA}) = 1$ without resorting to DPAs:

▶ **Theorem 14.** *The problem* $\mathbb{P}(\text{coNBA}) = 1$ *is PSPACE-complete.*

Theorem 11 (for NBAs) has a coNBA-analogue:

▶ **Theorem 15.** *The problem* $\mathbb{P}(\text{coNBA}) = 0$ *is EXPTIME-complete. It is EXPTIME-hard even for NBAs all whose states are accepting.*

5 Co-Unambiguous Büchi Automata

In this section we build on the previous section, in particular on Lemma 13, to derive our main technical result: given a BP $\mathcal{B}$ and an *unambiguous* Büchi automaton (UBA) $\mathcal{A}$, one can decide in NC whether $\mathcal{B}$ almost surely generates a tree all whose branches are rejected by $\mathcal{A}$:

▶ **Proposition 16.** *The problem* $\mathbb{P}(\text{coUBA}) = 1$ *is in NC.*

The rest of the section is devoted to the proof of this theorem. Fix a BP $\mathcal{B}$ and a UBA $\mathcal{A}$. Since NC is closed under complement, we can focus on the problem whether the probability is positive that a $\mathcal{B}$-tree has some branch accepted by $\mathcal{A}$. We use Lemma 12. Since reachability in a graph is in NL and, hence, in NC, it suffices to decide in NC whether the probability that some branch of a $\mathcal{B}[X_f]$-tree is accepted by $\mathcal{A}[f, X_f]$ is positive. By Lemma 13 it suffices to decide in NC whether the probability that some branch of a $\mathcal{B}_{det}$-tree does not have any nodes of type $\emptyset$ is positive. The challenge is that $\mathcal{B}_{det}$ may be exponentially larger than $\mathcal{A}$, so we need to exploit the unambiguousness of $\mathcal{A}$ and the regular structure it gives to $\mathcal{B}_{det}$.

Let $\mathcal{B}''_{det}$ be the BP (with ε-rules allowed) obtained from $\mathcal{B}_{det}$ by removing the type $\emptyset$ and eliminating all occurrences of type $\emptyset$ from all right-hand sides. The probability that a $\mathcal{B}_{det}$-tree has an infinite branch of non-$\emptyset$ nodes is equal to the probability that a $\mathcal{B}''_{det}$-tree is infinite. Hence, it remains to show that one can decide in NC whether the probability that a $\mathcal{B}''_{det}$-tree is infinite is positive.

Define a matrix $M \in \mathbb{Q}^{Q[f,X_f] \times Q[f,X_f]}$ whose rows and columns are indexed with the non-$\bar{q}_0$ states of $\mathcal{A}[f, X_f]$:

$$M_{(q,X),(r,Y)} := \begin{cases} \sum_{X \overset{p}{\hookrightarrow} u} p|u|_Y & \text{if } (q,X) \xrightarrow{Y} (r,Y) \text{ in } \mathcal{A}[f, X_f] \\ 0 & \text{otherwise}, \end{cases}$$

where $|u|_Y \in \mathbb{N}_0$ is the number of occurrences of Y in u. (Think of $M_{(q,X),(r,Y)}$ as the expected number of (r, Y)-"successors" of (q, X).) The graph of M is equal to the transition graph of $\mathcal{A}[f, X_f]$ (excluding $\bar{q}_0$), which is strongly connected.

Say that $\mathcal{A}[f, X_f]$ has *proper branching* if there exist $(q, Y) \xrightarrow{Z_1} (r_1, Z_1)$ and $(q, Y) \xrightarrow{Z_2} (r_2, Z_2)$ in $\mathcal{A}[f, X_f]$ and a rule $Y \overset{p}{\hookrightarrow} u_1 Z_1 u_2 Z_2 u_3$ in $\mathcal{B}$ with $u_1, u_2, u_3 \in \Gamma^*$. Now we can state the key lemma:

▶ **Lemma 17.** *Let ρ be the spectral radius of M. The probability that a $\mathcal{B}''_{det}$-tree is infinite is positive if and only if either $\rho > 1$ or $\rho = 1$ and $\mathcal{A}[f, X_f]$ does not have proper branching.*

Observe the similarity between Lemmas 5 and 17. In fact, the proof of Lemma 17, given below, is based on Lemma 5. Lemma 17 shows that properties of $\mathcal{A}[f, X_f]$ and M (which are polynomial-sized objects) determine a property of the exponential-sized BP $\mathcal{B}''_{det}$. Unambiguousness of $\mathcal{A}[f, X_f]$ is crucial for that connection.

Given that Lemma 17 reflects the key insight of this section (if not of this paper), let us comment further. Suppose $\mathcal{A}[f, X_f]$ has two outgoing transitions in a state (q, Y), say $(q, Y) \xrightarrow{Z_1} (r_1, Z_1)$ and $(q, Y) \xrightarrow{Z_2} (r_2, Z_2)$. This branching could be "proper branching" as defined before Lemma 17, or the original UBA $\mathcal{A}$ could be nondeterministic when reading Y in q and have transitions $q \xrightarrow{Y} r_1$ and $q \xrightarrow{Y} r_2$. Either type of branching causes non-0 entries in the matrix M and, intuitively, increases its spectral radius ρ. Lemma 17 tells us that the probability that a $\mathcal{B}''_{det}$-tree is infinite is governed by the *combined* effect on ρ of both types of branching: if $\rho > 1$ then a $\mathcal{B}''_{det}$-tree is infinite with positive probability; only in the borderline case, $\rho = 1$, the type of branching matters. Again, this characterization is only correct if the nondeterminism in $\mathcal{A}$ does not cause ambiguousness.

Let us consider what Lemma 17 states for the special case of Markov chains. In that case, clearly there is no proper branching. One can show, using unambiguousness, that for Markov chains the spectral radius ρ of the matrix M is at most 1. Hence, Lemma 17 states for Markov chains that the probability that a $\mathcal{B}''_{det}$-tree (consisting of a single branch) is infinite is positive if and only if $\rho = 1$. Indeed, a related statement can be found in [2, Lemma 6].

To finish the proof of Proposition 16 it suffices to show that we can check the conditions of Lemma 17 in NC. Indeed, for comparing the spectral radius with 1, we employ Lemma 3. One can check for proper branching in logarithmic space, hence in NC. This completes the proof of Proposition 16.

6 LTL

With Proposition 16 from the previous section, we can now show our headline result:

▶ **Theorem 18.** *The problem $\mathbb{P}(\text{LTL}) = 1$ is PSPACE-complete.*

Proof. PSPACE-hardness is immediate in two different ways. It follows both from the PSPACE-hardness of model checking Markov chains against LTL and from the PSPACE-hardness of model checking transition systems against LTL [31]. Both model-checking problems are special cases of $\mathbb{P}(\text{LTL}) = 1$.

Towards membership in PSPACE, there is a classical PSPACE procedure that translates an LTL formula into an (exponential-sized) Büchi automaton [37]. As noted by several authors (e.g., [12, 7]), this procedure can easily be adapted to ensure that the Büchi automaton be a UBA. By applying this translation to the negation $\neg\varphi$ of the input formula φ, we obtain a UBA that rejects exactly those words that satisfy φ. By Proposition 16 the problem $\mathbb{P}(\text{coUBA}) = 1$ is in NC. By Lemma 2 it follows that $\mathbb{P}(\text{LTL}) = 1$ is in PSPACE. ◀

Finally we show the following result, exhibiting a big complexity gap between the problems $\mathbb{P}(\text{LTL}) = 1$ and $\mathbb{P}(\text{LTL}) = 0$.

▶ **Theorem 19.** *The problem* $\mathbb{P}(\mathrm{LTL}) = 0$ *is 2EXPTIME-complete.*

Proof. For membership in 2EXPTIME, we use again the classical procedure that translates an LTL formula into an exponential-sized Büchi automaton [37] and then invoke Theorem 11.

For 2EXPTIME-hardness we adapt the reduction from [11, Theorem 3.2.1] for MDPs. The details are in [24]. ◀

7 Conclusions

We have devised a PSPACE procedure for $\mathbb{P}(\mathrm{LTL}) = 1$, i.e., qualitative LTL model checking of BPs. The best previously known procedure ran in 2EXPTIME [8]. Since BPs naturally generalize both transition systems and Markov chains (for both of which LTL model checking is PSPACE-complete), one might view our model-checking algorithm as an optimal general procedure. The same holds for NBA-specifications instead of LTL.

The main technical ingredients have been the automata-theoretic approach and the algorithmic analysis of UBAs, nonnegative matrices, and finiteness of BPs. Our proofs were inspired by the observation that the spectral radii of certain nonnegative matrices are central to model checking Markov chains against UBAs, and also determine fundamental properties of BPs. Very loosely speaking, when model checking Markov chains against UBAs, the spectral radius measures the amount of nondeterministic branching in the UBA, whereas when analyzing BPs, the spectral radius measures the amount of tree branching. The "general case", i.e., model checking BPs, features both kinds of branching. Serendipitously, an analysis of spectral radii still leads, as we have seen, to optimal algorithms.

We have also established the complexities of related problems, partially as a tool for the mentioned LTL and NBA problems and partially to map out the landscape. We have shown that the $\mathbb{P}(\cdot) = 0$ variants are more complex than their $\mathbb{P}(\cdot) = 1$ counterparts. An intuitive explanation of this phenomenon is that for an instance of an $\mathbb{P}(\cdot) = 1$ problems to be negative, tree branching and probabilistic branching "work together" to falsify the specification on some branch. In contrast, for $\mathbb{P}(\cdot) = 0$ problems, tree branching and probabilistic branching are "adversaries", like in MDPs. Indeed, for lower bounds on $\mathbb{P}(\cdot) = 0$ problems we have encoded alternation in various forms.

One might ask about the complexity of $\mathbb{P}(\mathrm{UBA}) = 1$. Indeed, in trying to solve $\mathbb{P}(\mathrm{LTL}) = 1$ efficiently, the authors set out to solve $\mathbb{P}(\mathrm{UBA}) = 1$ efficiently (perhaps in P or even NC), with the PSPACE transduction from LTL to UBA in mind. However, the complexity of UBA universality is an open problem [30]; only membership in PSPACE is known. So even for the fixed transition system with $a \overset{1}{\hookrightarrow} ab$ and $b \overset{1}{\hookrightarrow} ab$ the problem $\mathbb{P}(\mathrm{UBA}) = 1$ cannot be placed in P without improving the complexity of UBA universality. A PSPACE-hardness proof of $\mathbb{P}(\mathrm{UBA}) = 1$ might have to make use of both types of branching in BPs, as $\mathbb{P}(\mathrm{UBA}) = 1$ is in NC for Markov chains [2].

Model checking BPs quantitatively, i.e., computing the satisfaction probability, comparing it with a threshold, or approximating it, is left for future work. Exact versions of these problems are computationally complex, as they are at least as hard as the corresponding $\mathbb{P}(\cdot) = 0$ problem. The paper [8] describes, for DPAs, nonlinear equation systems whose least nonnegative solution characterizes the satisfaction probabilities. Newton's method is efficient for approximating the solution of such equation systems; see [33, 15].

References

1 K.B. Athreya and P.E. Ney. *Branching Processes*. Springer, 1972.

2 C. Baier, S. Kiefer, J. Klein, S. Klüppelholz, D. Müller, and J. Worrell. Markov chains and unambiguous Büchi automata. In *Proceedings of the 28th International Conference on Computer Aided Verification (CAV)*, volume 9779 of *LNCS*, pages 23–42, 2016.

3 A. Berman and R.J. Plemmons. *Nonnegative matrices in the mathematical sciences*. SIAM, 1994.

4 A. Borodin. On relating time and space to size and depth. *SIAM Journal of Computing*, 6(4):733–744, 1977. doi:10.1137/0206054.

5 A. Borodin, J. von zur Gathen, and J.E. Hopcroft. Fast parallel matrix and GCD computations. *Information and Control*, 52(3):241–256, 1982. doi:10.1016/S0019-9958(82)90766-5.

6 D. Bustan, S. Rubin, and M.Y. Vardi. Verifying omega-regular properties of Markov chains. In *Computer Aided Verification (CAV)*, volume 3114 of *Lecture Notes in Computer Science*, pages 189–201. Springer, 2004.

7 S. Chakraborty and J.-P. Katoen. Parametric LTL on Markov chains. In *8th IFIP TC 1/WG 2.2 International Conference on Theoretical Computer Science*, volume 8705 of *Lecture Notes in Computer Science*, pages 207–221. Springer, 2014.

8 T. Chen, K. Dräger, and S. Kiefer. Model checking stochastic branching processes. In *Mathematical Foundations of Computer Science (MFCS)*, volume 7464 of *Lecture Notes in Computer Science*, pages 271–282. Springer, 2012.

9 R. Cleaveland, S.P. Iyer, and M. Narasimha. Probabilistic temporal logics via the modal mu-calculus. *Theoretical Computer Science*, 342(2-3):316–350, 2005.

10 C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *Symposium on Foundations of Computer Science (FOCS)*, pages 338–345. IEEE Computer Society, 1988.

11 C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

12 J.-M. Couvreur, N. Saheb, and G. Sutre. An optimal automata approach to LTL model checking of probabilistic systems. In *International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR)*, volume 2850 of *Lecture Notes in Computer Science*, pages 361–375. Springer, 2003.

13 J. Esparza, A. Gaiser, and S. Kiefer. A strongly polynomial algorithm for criticality of branching processes and consistency of stochastic context-free grammars. *Information Processing Letters*, 113(10-11):381–385, 2013.

14 K. Etessami, E. Martinov, A. Stewart, and M. Yannakakis. Reachability for branching concurrent stochastic games. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132 of *LIPIcs*, pages 115:1–115:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

15 K. Etessami, A. Stewart, and M. Yannakakis. A polynomial time algorithm for computing extinction probabilities of multitype branching processes. *SIAM Journal of Computing*, 46(5):1515–1553, 2017.

16 K. Etessami, A. Stewart, and M. Yannakakis. Greatest fixed points of probabilistic min/max polynomial equations, and reachability for branching Markov decision processes. *Information and Computation*, 261:355–382, 2018.

17 K. Etessami, A. Stewart, and M. Yannakakis. Polynomial time algorithms for branching Markov decision processes and probabilistic min(max) polynomial Bellman equations. *Mathematics of Operations Research*, 45(1):34–62, 2020.

18 K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the ACM*, 56(1):1:1–1:66, 2009.

19 K. Etessami and M. Yannakakis. Model checking of recursive probabilistic systems. *ACM Transactions on Computational Logic*, 13(2):12:1–12:40, 2012. doi:10.1145/2159531.2159534.

20 T. Gogacz, H. Michalewski, M. Mio, and M. Skrzypczak. Measure properties of regular sets of trees. *Information and Computation*, 256:108–130, 2017.

21 A. Gorlin and C.R. Ramakrishnan. Separable GPL: decidable model checking with more non-determinism. In *International Conference on Concurrency Theory (CONCUR)*, volume 118 of *LIPIcs*, pages 36:1–36:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

22 P. Haccou, P. Jagers, and V.A. Vatutin. *Branching Processes: Variation, Growth, and Extinction of Populations.* Cambridge University Press, 2005.

23 T.E. Harris. *The Theory of Branching Processes.* Springer, 1963.

24 S. Kiefer, P. Semukhin, and C. Widdershoven. Linear-time model checking branching processes. Technical report, arxiv.org, 2021. Available at `arXiv:2107.01687`.

25 O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Principles of Programming Languages (POPL)*, pages 97–107. ACM Press, 1985.

26 H. Michalewski and M. Mio. On the problem of computing the probability of regular sets of trees. In *Foundation of Software Technology and Theoretical Computer Science (FSTTCS)*, volume 45 of *LIPIcs*, pages 489–502. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015.

27 D. Niwiński, M. Przybyłko, and M. Skrzypczak. Computing measures of weak-MSO definable sets of trees. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 168 of *LIPIcs*, pages 136:1–136:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

28 C.M. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

29 N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Logical Methods Computer Science*, 3(3), 2007. `doi:10.2168/LMCS-3(3:5)2007`.

30 A. Rabinovich. Complementation of finitely ambiguous Büchi automata. In *Developments in Language Theory (DLT)*, volume 11088 of *Lecture Notes in Computer Science*, pages 541–552. Springer, 2018.

31 A.P Sistla and E.M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985. `doi:10.1145/3828.3837`.

32 A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with appplications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987. `doi:10.1016/0304-3975(87)90008-9`.

33 A. Stewart, K. Etessami, and M. Yannakakis. Upper bounds for Newton's method on monotone polynomial systems, and P-time model checking of probabilistic one-counter automata. *Journal of the ACM*, 62(4):30:1–30:33, 2015.

34 W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. B): Formal Models and Semantics*, chapter 4, pages 133–191. MIT Press, 1990.

35 M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 327–338. IEEE Computer Society, 1985. `doi:10.1109/SFCS.1985.12`.

36 M.Y. Vardi. Probabilistic linear-time model checking: An overview of the automata-theoretic approach. In *Formal Methods for Real-Time and Probabilistic Systems*, pages 265–276. Springer, 1999.

37 M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *1st Symposium on Logic in Computer Science (LICS)*, pages 332–344. IEEE Computer Society Press, 1986.

Quantified Linear Temporal Logic over Probabilistic Systems with an Application to Vacuity Checking

Jakob Piribauer
Technische Universität Dresden, Germany

Christel Baier
Technische Universität Dresden, Germany

Nathalie Bertrand
Université Rennes, Inria, CNRS, IRISA, France

Ocan Sankur
Université Rennes, Inria, CNRS, IRISA, France

Abstract

Quantified linear temporal logic (QLTL) is an ω-regular extension of LTL allowing quantification over propositional variables. We study the model checking problem of QLTL-formulas over Markov chains and Markov decision processes (MDPs) with respect to the number of quantifier alternations of formulas in prenex normal form. For formulas with $k-1$ quantifier alternations, we prove that all qualitative and quantitative model checking problems are k-EXPSPACE-complete over Markov chains and $k+1$-EXPTIME-complete over MDPs.

As an application of these results, we generalize vacuity checking for LTL specifications from the non-probabilistic to the probabilistic setting. We show how to check whether an LTL-formula is affected by a subformula, and also study inherent vacuity for probabilistic systems.

2012 ACM Subject Classification Theory of computation → Verification by model checking

Keywords and phrases Quantified linear temporal logic, Markov chain, Markov decision process, vacuity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.7

Related Version *Extended Version*: `https://wwwtcs.inf.tu-dresden.de/ALGI/PUB/CONCUR21/` [19]

Funding This work was funded by DFG grant 389792660 as part of TRR 248 – CPEC (see `https://perspicuous-computing.science`), the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy), DFG-projects BA-1679/11-1 and BA-1679/12-1, and the Research Training Group QuantLA (GRK 1763).

1 Introduction

In the formal verification of probabilistic systems, a central problem is the *model-checking problem*: Given a system model $\mathcal{M}$ and a specification φ, decide whether the probability $\Pr_{\mathcal{M}}(\varphi)$ that φ holds on an execution of $\mathcal{M}$ is 1 or whether it is positive, respectively, (qualitative model checking) or compute the probability $\Pr_{\mathcal{M}}(\varphi)$ (quantitative model checking). In case the system exhibits non-deterministic behavior, the model-checking problems address the worst- or best-case resolution of the non-determinism, i.e., the minimal or maximal satisfaction probability among all possible resolutions of the non-deterministic choices. Common probabilistic system models are finite-sate Markov chains that are purely probabilistic and Markov decision processes (MDPs) that also model non-deterministic behavior. Specifications can be formulated in temporal logics, such as linear temporal logic (LTL) as an important example, or be given by automata, such as non-deterministic Büchi automata (NBA). The choice of the specification formalism is a balancing act between expressive power, succinctness,

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 7; pp. 7:1–7:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and the complexity of the respective model-checking problems. Additionally, the formalism should allow one to describe desired system behaviors in a way comprehensible to a human user as writing down the specification is itself an error-prone process in practice.

Quantified linear temporal logic (QLTL), introduced by Sistla [21], is an extension of LTL with quantification over propositional variables lifting the expressive power from star-free to all ω-regular languages. A formula of the form $\exists x.\varphi$ holds on a word w if one can choose a set of positions at which x holds such that the word w extended with this choice satisfies φ. The quantification hence ranges over all sets of positions, i. e., sets of natural numbers. In QLTL, LTL-formulas can be extended with the quantification over propositions that, for example, capture hidden variables or encode annotations of a trace. This can be useful if we want to define properties not expressible in LTL in a context in which one often works with LTL. Examples include definitions of refinement relations in which internal variables are quantified to express equivalence of two specifications with respect to the observable variables [14], a necessary and sufficient condition expressed as a QLTL-formula on the serializability of histories in concurrent database scheduling produced by a scheduler whose behavior is expressed by an LTL-formula [13], or the QLTL-expressible existence of finite counterexamples witnessing the unrealizability of an LTL-specification for distributed fault-tolerant systems [8]. Furthermore, the vacuous satisfaction of a specification in a transitions system indicating that parts of the specification are irrelevant for the satisfaction has been defined using QLTL [1]. We transfer this definition of vacuous satisfaction to the probabilistic setting in this paper and explain the notion of vacuity in more detail below. In all of these successful applications of QLTL to questions in formal verification, the necessary QLTL-formulas require only few quantifier alternations; often even a single block of quantifiers without alternation is sufficient.

The full logic, however, is not suitable for practical applications: the non-probabilistic model-checking problem of QLTL on transition systems has non-elementary complexity [22]. The lower bounds can be pinpointed to the different levels of quantifier alternation of formulas in prenex normal form. Model-checking of QLTL-formulas with $k-1$ quantifier alternations in transition systems is k-EXPSPACE-complete. Distinguishing whether the first block of quantifiers is existential (Σ_k^{QLTL}) or universal (Π_k^{QLTL}) refines the result as for Π_k^{QLTL}-formulas the complexity of model-checking drops to $k{-}1$-EXPSPACE-completeness [22]. The increase of the complexity by one exponential per quantifier alternation is theoretically intriguing on the one hand, and on the other hand leads to reasonable complexity results for properties that can be expressed succinctly with the use of few quantifier alternations. A similar complexity hierarchy is observed in other settings. The complexity of model checking quantified computation tree logic (CTL) with k quantifier alternations is k-EXPTIME-complete, and it is $k+1$-EXPTIME-complete for quantified CTL* in the tree semantics; while in the structure semantics, these problems span the polynomial hierarchy [17]. The hardness of the fragments of QLTL [22] was used to show that model checking strategy logic is k-EXPSPACE-hard when restricted to k quantifier alternations.

In this paper, we study the model-checking problem of QLTL in probabilistic systems. Our main result is that the complexity of the model-checking problems on Markov chains and MDPs match the upper bounds obtained via straight-forward automata constructions: For Markov chains and Σ_k^{QLTL}- and Π_k^{QLTL}-formulas, all model-checking problems are k-EXPSPACE-complete, while for MDPs the problems are $k+1$-EXPTIME-complete. These complexity results are summarized in Table 1. As the upper bounds are easily obtained, the main contribution lies in proving the lower bounds. The hardness proofs for Markov chains, encode a tiling problem of a k-exponentially wide rectangle with arbitrary height. For the hardness proofs for MDPs, we encode the computation of an alternating k-exponentially space-bounded Turing machine. The alternation can be mimicked in an MDP by letting one player in the acceptance game of the alternating Turing machine be played randomly, while the scheduler takes the role of the other player. We obtain the result that the complexities of the model-checking problems for Σ_k^{QLTL} and Π_k^{QLTL} coincide in the probabilistic setting in contrast to the asymmetry known for the non-probabilistic setting. It is remarkable that the complexities of Σ_1^{QLTL}- and Π_1^{QLTL}-model checking in MDPs are the same as the complexity of LTL-model checking. For each further quantifier alternation, the complexity increases by one exponential. In contrast, we see an exponential increase in complexity already for the first block of quantifiers in Σ_1^{QLTL} and Π_1^{QLTL} compared to LTL-model checking in Markov chains.

Table 1 Complexity results for the model-checking problems of fragments of QLTL. All entries state completeness results.

	transition system	Markov chain	MDP
LTL	PSPACE [23, 25]	PSPACE [7]	2-EXPTIME [7]
Π_1^{QLTL}	PSPACE [22]	**EXPSPACE**	**2-EXPTIME**
Σ_1^{QLTL}	EXPSPACE [22]	**EXPSPACE**	**2-EXPTIME**
Π_k^{QLTL}	k-1-EXPSPACE [22]	**k-EXPSPACE**	**k+1-EXPTIME**
Σ_k^{QLTL}	k-EXPSPACE [22]	**k-EXPSPACE**	**k+1-EXPTIME**

On the one hand, knowledge of the precise complexities of the model-checking problems for Σ_k^{QLTL}- and Π_k^{QLTL}-formulas over probabilistic systems might be useful to determine the complexity of other problems in the formal verification of probabilistic systems – in particular, by using the new lower bounds provided in this paper for new hardness results. On the other hand, the upper bounds are obtained via the construction of automata. It follows easily that all investigated model-checking problems can be solved in time polynomial in the size of the model, i.e., the Markov chain or the MDP. This means that efficient model checking for low levels of the quantifier alternation hierarchy of QLTL might be possible in many application areas despite the high complexities of the model-checking problems because formulas are typically small compared to the size of the models.

As an application of our main results, we extend the definition of vacuous satisfaction of a specification from [1] to the probabilistic setting. For an illustration of vacuous satisfaction, consider the specification: "Whenever a request is sent, it is eventually granted." If in a system model no requests are ever sent, the specification is satisfied and a model checker would report this result. However, something is obviously wrong with either the specification or – in this case more likely – the system model. We say the specification is vacuously true. The formal definition of vacuity that we generalize to the probabilistic setting captures the fact that the truth values of the grants in the specification do not influence the satisfaction of the specification at all. We could replace "it is eventually granted" with any arbitrary requirement or even choose an arbitrary set of positions at which that part of the specification should be true and the specification would still hold in the system model. We say that this subformula does not *affect* the satisfaction of the specification. Perturbing the truth values in arbitrary ways is expressed by a universal quantification over a proposition in the formal definition. A vacuity check during the model checking process can be an invaluable help as it can detect such severe errors in the design of the model or the specification in an early stage of the development that would otherwise stay undetected if the model checker returns the desired result.

We provide a generalization of the definition of affection that is suitable for the probabilistic setting. We prove that Π_1^{QLTL}-model checking is inter-reducible with the question whether a subformula affects a formula in a probabilistic system. Hence, an additional vacuity check according to this definition does not increase the complexity of model checking in MDPs. For Markov chains, however, the additional vacuity check would lead to an exponential blow-up of the procedure as shown by our new lower bound for Π_1^{QLTL}-model checking over Markov chains. Consequently, we turn our attention to the notion of inherent vacuity introduced in [9]. This notion captures that a specification is vacuous, i.e. not affected by some subformula, in all models. So, while disregarding the interplay between model and specification, inherent vacuity indicates a severe error in the specification. For all natural variants of this definition for Markov chains and MDPs that can be obtained using our notion of affection, we obtain the result that inherent vacuity of a specification can be checked by a (non-probabilistic) validity check of a Π_1^{QLTL}-formula. Therefore, inherent vacuity for Markov chains and MDPs can be checked in polynomial space rendering the addition of a check for inherent vacuity to the model checking procedure potentially useful and reasonable in practice.

Related Work

Closest to our main complexity hierarchy result is the complexity hierarchy result for QLTL in the non-probabilistic setting [22]. Over probabilistic systems, the model-checking problems for Wolper's ETL [26], another ω-regular extension of LTL, which uses automata operators, is investigated in [7] and shown to lie in EXPTIME. We are not aware of any explicit investigations of QLTL or further ω-regular extensions of LTL, such as Gabbay's USF [10], an extension with fixed-point operators, over probabilistic systems.

Concerning vacuity checking, various notions have been studied for non-probabilistic systems. In [3] and [16], a notion of *formula vacuity* for fragments of CTL* is investigated in which the underlying notion of non-affection means that a subformula can be replaced by any other formula without affecting the truth of the formula in a model. *Trace vacuity* for LTL, which we generalize to the probabilistic setting, was introduced in [1]. The authors argue that trace vacuity has advantages over formula vacuity as it is more robust with respect to changes of the model or the specification language. Based on this definition, the notion of inherent vacuity, which we adapt to the probabilistic setting, was introduced in [9]. Trace vacuity has been extended to various other logics such as CTL* [11] relying on a propositionally quantified version of the logic, or to the logic RELTL, an extension of LTL with regular layers, by universally quantifying interval variables [4]. In [12], a variety of degrees to which a formula can be vacuous is defined and analyzed in the context of CTL-model checking. For a survey covering different approaches of vacuity checking, we refer the reader to [15].

2 Preliminaries

We suppose familiarity with basic concepts of discrete Markovian models, LTL, and ω-automata, and only provide a brief summary of the notions and our notation. Details can be found in textbooks, e.g., [2, 6, 20]. Furthermore, we provide definitions regarding QLTL and state basic results.

2.1 Basic definitions

Markov decision processes (MDPs)

An MDP is a tuple $\mathcal{M} = (S, \mathit{Act}, P, s_{\mathit{init}}, \mathsf{AP}, L)$ where S is a finite state space, Act a finite set of actions, $P : S \times \mathit{Act} \times S \to [0,1] \cap \mathbb{Q}$ the transition probability function satisfying $\sum_{t \in S} P(s, \alpha, t) \in \{0,1\}$ for all $(s, \alpha) \in S \times \mathit{Act}$, $s_{\mathit{init}} \in S$ the initial state, AP a finite set of atomic propositions, and $L : S \to 2^{\mathsf{AP}}$ a labeling function. The triples (s, α, t) with $P(s, \alpha, t) > 0$ are called transitions of $\mathcal{M}$. The actions enabled in s form $\mathit{Act}(s) = \{\alpha \in \mathit{Act} : \sum_{t \in S} P(s, \alpha, t) = 1\}$. The *size* of an MDP is the number of states and actions plus the sum of the logarithmic lengths of the transition probabilities. Intuitively, when $\mathcal{M}$ is at a state s, then an action α of $\mathit{Act}(s)$ is selected nondeterministically; afterwards the next state is obtained by probabilistically choosing one of the potential successor states according to the probability distribution $P(s, \alpha, \cdot)$. Paths in MDP are alternating sequences of states and actions: $\pi = s_0\, \alpha_0\, s_1\, \alpha_1 \ldots$ where $\alpha_i \in \mathit{Act}(s_i)$ and $P(s_i, \alpha_i, s_{i+1}) > 0$ for all $i \geqslant 0$. We write $\pi_{[i\ldots]}$ for the suffix starting from s_i. The *trace* of π is the word $L(\pi) = L(s_0)\, L(s_1)\, L(s_2) \ldots$ over 2^{AP} obtained by projecting states to their labels. We do not distinguish between a path and its trace when the intended meaning is clear from context. A *scheduler* for $\mathcal{M}$ is a function $\mathfrak{S}$ that maps a finite path ζ to a probability distribution over $\mathit{Act}(\mathit{last}(\zeta))$ where $\mathit{last}(\zeta)$ is the last state of ζ. The function $\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M},s}$ denotes the probability measure induced by $\mathfrak{S}$, when s is the initial state. It is well-known that all ω-regular path properties φ are measurable and there exist schedulers maximizing or minimizing the probability for φ (see, e.g., [2]). This justifies the notations $\mathrm{Pr}^{\max}_{\mathcal{M},s}(\varphi) = \max_{\mathfrak{S}} \mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M},s}(\varphi)$ and analogously $\mathrm{Pr}^{\min}_{\mathcal{M},s}(\varphi)$ for ω-regular properties.

A *Markov chain* is a tuple $\mathcal{M} = (S, P, s_{\mathit{init}}, \mathsf{AP}, L)$ which can be seen as an MDP with only one action. The transition probability function $P : S \times S \to [0,1] \cap \mathbb{Q}$ does not include the action anymore and satisfies $\sum_{t \in S} P(s,t) \in \{0,1\}$ for all $s \in S$. There are no non-deterministic choices and $\mathrm{Pr}_{\mathcal{M},s}$ denotes the induced probability measure on maximal paths starting in s.

ω-automata

A *non-deterministic Büchi automaton (NBA)* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, Q_0, F)$ where Q is a finite set of states, Σ an alphabet, $\delta \subseteq S \times \Sigma \times S$ the transition relation, $Q_0 \subseteq Q$ the set of initial states and $F \subseteq Q$ the set of final states. A word $w = w_0\, w_1\, \ldots$ in Σ^ω is accepted by $\mathcal{A}$ if there is a run $q_0\, w_0\, q_1\, w_1\, q_2\, \ldots$ such that $q_0 \in Q_0$, $(q_i, w_i, q_{i+1}) \in \delta$ for all i, and for infinitely many i, $q_i \in F$. The language $\mathcal{L}(\mathcal{A})$ is the set of words accepted by $\mathcal{A}$.

2.2 Quantified linear temporal logic (QLTL)

Let AP be a finite set of atomic propositions. The syntax of linear temporal logic (LTL) is given by

$$\varphi ::= a | \varphi \wedge \varphi | \neg \varphi | \bigcirc \varphi | \varphi \,\mathrm{U}\, \varphi$$

where $a \in \mathsf{AP}$. The semantics is given on words in $(2^{\mathsf{AP}})^\omega$: For a word $w = w_0, w_1, \ldots$, we have $w \vDash a$ if $a \in w_0$; $w \vDash \bigcirc\varphi$ if $w_1, w_2, \dots \vDash \varphi$; and $w \vDash \varphi \,\mathrm{U}\, \psi$ if there is a $j \in \mathbb{N}$ such that $w_j, w_{j+1}, \dots \vDash \psi$ and $w_i, w_{i+1}, \dots \vDash \varphi$ for all $i < j$. The semantics of the Boolean connectives is defined as usual. For more details, consult, e.g., [2]. The logic QLTL is an extension of LTL with quantification over atomic propositions. We extend the syntax of LTL by allowing existential quantification $\exists x.\varphi$ over additional, fresh atomic propositions $x \notin \mathsf{AP}$ where φ is an LTL-formula over $\mathsf{AP} \cup \{x\}$. We further allow the common abbreviations $\top$ for true, $\perp$ for false, $\vee$, $\rightarrow$, $\leftrightarrow$, $\Diamond$, $\Box$, and $\forall x$. For a word $w \in (2^{\mathsf{AP}})^\omega$, we define that $w \vDash \exists x.\varphi$ if and only if there is a set $X \subseteq \mathbb{N}$ such that the word w' with $w'[i] = w[i]$ if $i \notin X$ and $w'[i] = w[i] \cup \{x\}$ if $i \in X$ satisfies $w' \vDash \varphi$. Consider the following example to illustrate the semantics of QLTL:

$$\begin{array}{llllll} \{a\} & \{b\} & \{a\} & \{c\} & \dots & \vDash \exists x.\Box(x \leftrightarrow \neg a) \text{ because} \\ \{a\} & \{b,x\} & \{a\} & \{c,x\} & \dots & \vDash \Box(x \leftrightarrow \neg a). \end{array}$$

For a QLTL-formula ϑ over AP, we allow arbitrarily many additional atomic propositions but require that all atomic propositions not in AP are quantified. We distinguish QLTL-formulas in prenex normal form according to the number of quantifier alternations. For $k \geqslant 1$, let Σ_k^{QLTL} be the set of QLTL-formulas of the form

$$\underbrace{\exists^*\forall^*\exists^*\dots}_{k \text{ blocks of quantifiers}} \varphi \equiv \underbrace{\exists^*\neg\exists^*\neg\exists^*\dots}_{k \text{ blocks of quantifiers}} (\neg)\varphi$$

where φ is quantifier-free, i.e. an LTL-formula. Likewise, let Π_k^{QLTL} be the set of QLTL-formulas starting with k blocks of quantifiers followed by a quantifier-free formula such that the first block is $\forall^*$. The negation of a Σ_k^{QLTL}-formula is equivalent to a Π_k^{QLTL}-formula.

QLTL, and in particular Σ_1^{QLTL}, can express exactly all ω-regular properties. In fact, the existence of an accepting run on a word in an NBA $\mathcal{A}$ with states Q can be expressed by a Σ_1^{QLTL}-formula with $|Q|$-many existential quantifications followed by an LTL-formula [21].

Conversely, for a Σ_k^{QLTL}-formula $\vartheta = \exists^*\neg\exists^*\dots(\neg)\varphi$, we can build an NBA of k-exponential size accepting exactly the words satisfying ϑ: For the LTL-part $(\neg)\varphi$, we first construct an NBA of exponential size (see [25]). Existential quantification on the NBA-level is easy as it corresponds to standard projection onto the non-quantified variables; a quantified variable x is simply removed from the labels of the transition relation. This introduces new non-deterministic choices between the options to take a transition requiring a letter, i.e. a set of atomic propositions, P or a transition requiring $P \cup \{x\}$ when reading P. The quantifier prefix contains $k-1$ negations in addition to the existential quantifiers. Each of these negations requires a complementation of the automaton constructed so far before we can use projection again to account for the next block of quantifiers. Each complementation increases the size by one further exponential. Hence, the procedure produces an NBA for ϑ of k-exponential size in k-exponential time (see [22] for more details).

3 QLTL model checking in probabilistic systems

This section is devoted to proving the complexity hierarchy results in terms of the quantifier alternation for the model-checking problem of QLTL in probabilistic systems. More precisely, our goal is to pinpoint the complexities of the following problems, for Π_k^{QLTL}- or Σ_k^{QLTL}-formulas φ:

- Qualitative model-checking problems:
  - For a Markov chain $\mathcal{M}$, decide whether $\Pr_{\mathcal{M},s_{init}}(\varphi) = 1$, or whether $\Pr_{\mathcal{M},s_{init}}(\varphi) > 0$, respectively.
  - For an MDP $\mathcal{M}$, decide whether $\Pr^{\max}_{\mathcal{M},s_{init}}(\varphi) = 1$, whether $\Pr^{\max}_{\mathcal{M},s_{init}}(\varphi) > 0$, whether $\Pr^{\min}_{\mathcal{M},s_{init}}(\varphi) = 1$, or whether $\Pr^{\min}_{\mathcal{M},s_{init}}(\varphi) > 0$, respectively.
- Quantitative model-checking problems:
  - For a Markov chain $\mathcal{M}$, compute $\Pr_{\mathcal{M},s_{init}}(\varphi)$. For hardness results, we consider the decision versions whether $\Pr_{\mathcal{M},s_{init}}(\varphi) \bowtie \vartheta$ for a given $\vartheta \in \mathbb{Q}$ and $\bowtie \in \{\leqslant, <, >, \geqslant\}$.

- For an MDP $\mathcal{M}$, compute $\mathrm{Pr}^{\mathrm{opt}}_{\mathcal{M},s_{init}}(\varphi)$ for opt $\in \{\max, \min\}$. For hardness results, we consider the decision versions whether $\mathrm{Pr}^{\mathrm{opt}}_{\mathcal{M},s_{init}}(\varphi) \bowtie \vartheta$ for a given $\vartheta \in \mathbb{Q}$, $\bowtie \in \{\leqslant, <, >, \geqslant\}$, and opt $\in \{\max, \min\}$.

We restrict our attention to QLTL-formulas in prenex normal form. While we have seen that all QLTL-formulas are equivalent to a Σ_1^{QLTL}-formula, the transformation from arbitrary QLTL-formulas to Σ_1^{QLTL}-formulas has non-elementary complexity. The lower bound for this transformation is a direct consequence of the complexity hierarchy result for the non-probabilistic model-checking problem mentioned above. However, there is a polynomial-time transformation to prenex normal form for QLTL-formulas: After renaming all quantified variables such that each quantifier quantifies a unique variable not occurring outside the scope of this quantifier, we can pull out quantifiers using the following equivalences for arbitrary QLTL-formula φ and ψ where ψ does not contain the atomic proposition x and both formulas do not contain t:

1. $(\exists x \varphi) \,\mathrm{U}\, \psi \equiv \forall t \exists x((t \,\mathrm{U}(\neg t \wedge \varphi)) \vee (t \,\mathrm{U}\, \psi))$.
2. $(\forall x \varphi) \,\mathrm{U}\, \psi \equiv \forall x(\varphi \,\mathrm{U}\, \psi)$.
3. $\psi \,\mathrm{U}(\exists x \varphi) \equiv \exists x(\psi \,\mathrm{U}\, \varphi)$.
4. $\psi \,\mathrm{U}(\forall x \varphi) \equiv \exists t \forall x((\psi \wedge t) \,\mathrm{U}(\varphi \wedge \neg t))$.

Note that in the first and last equivalence where t is quantified, only the first position where $\neg t$ holds is important for the subsequent formulas. In this way, the quantification over t corresponds to the quantification over positions in the semantics of the U-operator. For $Q \in \{\exists, \forall\}$, we further have $\bigcirc Q x \varphi \equiv Q x \bigcirc \varphi$ and moving quantifiers to the front over Boolean connectives can be done as usual. So, we can transform a QLTL-formula to prenex normal form in polynomial time while introducing new quantifiers to account for the implicit quantification over positions of the U-operator.

In applications of QLTL in formal verification, however, quantified variables are mostly used to describe possible annotations of a trace or traces of hidden variables. Hence, the quantified traces are supposed to be constant once chosen and not to be reassigned when evaluating subformulas on different suffixes. Thus, these formulas often are already in prenex normal form.

Our main result concerning QLTL-model checking over probabilistic systems is the following complexity hierarchy result:

▶ **Theorem 1** (Main Result). *All qualitative and quantitative model-checking problems for Σ_k^{QLTL} and Π_k^{QLTL} in Markov chains are k-EXPSPACE-complete and can be solved in time polynomial in the size of the Markov chain.*

All qualitative and quantitative model-checking problems for Σ_k^{QLTL} and Π_k^{QLTL} with $k \geqslant 1$ in MDPs are $k+1$-EXPTIME-complete and can be solved in time polynomial in the size of the MDP.

The upper bounds are obtained by the straight-forward construction of NBAs as described above (Section 2.2). The main contribution hence is the proof of the lower bounds. For Markov chains, we provide a reduction from a tiling problem that simultaneously shows hardness for all qualitative model-checking problems (Theorem 2). We afterwards conclude the same complexity result for all quantitative model-checking problems (Corollary 3). For MDPs, the result requires two different hardness proofs (Theorem 4): The hardness results for model-checking problems regarding the maximal satisfaction probability of Π_k^{QLTL}-formulas (or analogously the minimal satisfaction probability of Σ_k^{QLTL}-formulas) are somewhat simpler. We encode computations of an alternating Turing machine that is k-exponentially space bounded and can directly use sequences of k-exponentially many extended tape symbols for the encoding. For the hardness proof concerning the minimal satisfaction probability of Π_k^{QLTL}-formulas, we have to include a binary counter of $k-1$-exponential length separating two successive tape symbols in the encoding. In the hardness proof for Markov chains, we use a similar counter. So, the final hardness proof combines the ideas behind the first hardness proof for MDPs and the hardness proof for Markov chains. The same complexity results for all quantitative model-checking problems in MDPs can be concluded afterwards (Corollary 5).

3.1 Markov chains

We first address the qualitative model-checking problems in Markov chains. We provide a proof sketch for the hardness proof. The full proof can be found in [19].

▶ **Theorem 2.** *For any k, all qualitative model-checking problems for Σ_k^{QLTL} and Π_k^{QLTL} in Markov chains are k-EXPSPACE-complete and can be solved in time polynomial in the size of the Markov chain.*

Proof sketch. The upper bounds are obtained by building NBAs of k-exponential size for Σ_k^{QLTL}-formulas as described in Section 2. The negation of a Π_k^{QLTL}-formula is equivalent to a Σ_k^{QLTL}-formula of the same length. As all qualitative model-checking problems for NBAs in Markov chains are PSPACE-complete and can be solved in time polynomial in the size of the Markov chain [7], we obtain the upper bounds.

For the hardness results, we use a reduction from k-exponential tiling problems. We define the following function $h\colon \mathbb{N}^2 \to \mathbb{N}$: Let $h(0,n) = n$ for all n and $h(k+1,n) = 2^{h(k,n)} \cdot h(k,n)$ for all k. So, $h(k,n)$ is k-exponential in n. The following k-exponential tiling problem is known to be k-EXPSPACE-complete [24]:

Given: a finite set of tiles T, two relations $H \subseteq T^2$ and $V \subseteq T^2$, an initial tile $t_0 \in T$ and a final tile $t_f \in T$ as well as a natural number n in unary.

Question: Is there an $m \in \mathbb{N}$ such that the $2^{h(k-1,n)} \times (m+1)$-grid $\{0,\dots,2^{h(k-1,n)}-1\} \times \{0,\dots,m\}$ can be tiled, i. e., is there a function $f\colon \{0,\dots,2^{h(k-1,n)}-1\} \times \{0,\dots,m\} \to T$, such that:

1. the tile at position $(0,0)$ is the initial tile t_0 and the tile at position $(0,m)$ is the final tile t_f; in other words, $f(0,0) = t_0$ and $f(0,m) = t_f$,
2. two tiles placed next to each other horizontally satisfy the relation H; more precisely, for any $0 \leqslant i < 2^{h(k-1,n)} - 1$ and $0 \leqslant j \leqslant m$, the pair $(f(i,j), f(i+1,j)) \in H$, and
3. two tiles placed next to each other vertically satisfy the relation V; more precisely, for any $0 \leqslant i \leqslant 2^{h(k-1,n)} - 1$ and $0 \leqslant j < m$, the pair $(f(i,j), f(i,j+1)) \in V$?

Given an instance of the k-exponential tiling problem, we construct a Markov chain $\mathcal{M}$ and a ψ in Π_k^{QLTL} such that $\Pr_{\mathcal{M}}(\psi) = 1$ iff $\Pr_{\mathcal{M}}(\psi) > 0$ iff there is a valid tiling. This establishes k-EXPSPACE-hardness for both qualitative model checking problems for Π_k^{QLTL}. As the negation of ψ is in Σ_k^{QLTL} and k-EXPSPACE is closed under complementation, the same result holds for Σ_k^{QLTL}.

Let $T = \{t_0, \dots, t_\ell\}$ be the set of tiles that we also use as atomic propositions and let $\{\mathit{start}, \mathit{end}, 0, 1\}$ be further atomic propositions. We construct a simple Markov chain $\mathcal{M}$, depicted in Figure 1, that almost surely produces a concatenation of infinitely many words from $\mathit{start}(T \cup \{0,1\})^+\mathit{end}$ that contains each of the finite words in $\mathit{start}(T \cup \{0,1\})^+\mathit{end}$. Some of these finite words will encode potential tilings. Namely, we encode a function $f\colon \{0,\dots,2^{h(k-1,n)}-1\} \times \{0,\dots,m\} \to T$ in the word

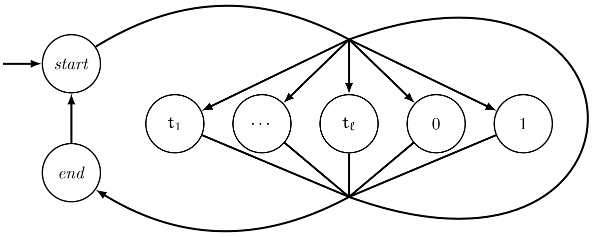


Figure 1 The Markov chain $\mathcal{M}$.

$$\begin{aligned} &\textit{start},\ f(0,0),\ \overbrace{0,0,0,\dots,0}^{h(k-1,n)\text{ steps}}, f(1,0),\ \overbrace{1,0,0,\dots,0}^{h(k-1,n)\text{ steps}},\dots,f(2^{h(k-1,n)}-1,0),\ \overbrace{1,1,1,\dots,1}^{h(k-1,n)\text{ steps}},\\ &\qquad f(0,1),\ \dots,\\ &\qquad f(0,m),\ \overbrace{0,0,0,\dots,0}^{h(k-1,n)\text{ steps}}, f(1,m),\ \overbrace{1,0,0,\dots,0}^{h(k-1,n)\text{ steps}},\dots,f(2^{h(k-1,n)}-1,m),\ \overbrace{1,1,1,\dots,1}^{h(k-1,n)\text{ steps}}, \textit{end}. \end{aligned}$$

For a valid encoding, the blocks of $h(k-1,n)$ bits have to encode a correct binary counter modulo $2^{h(k-1,n)}$, where the first bit is the least significant one, starting with $0\dots0$ after *start* and ending in $1\dots1$ before *end*. The encoding of the counter makes sure that indeed a function from a rectangle $\{0,\dots,2^{h(k-1,n)}-1\}\times\{0,\dots,m\}$ for some m is encoded.

Further, we construct a Π_k^{QLTL}-formula *valid_tiling* that expresses that at some point a valid tiling is encoded in a run. Several of the conditions including the initial, final and horizontal condition can easily be expressed. As tiles that are vertically adjacent in a tiling are separated by $h(k,n)=h(k-1,n)\cdot 2^{h(k-1,n)}$ steps, however, we have to employ additional ideas to express that all conditions on a valid encoding of a valid tiling are satisfied at some point. An important ingredient for our reduction is the collection of $\Sigma_{k-1}^{\mathrm{QLTL}}$-formulas $\varphi_{k-1,n}(p,q)$ from [22]. For each n and k from $\mathbb{N}$, the formula $\varphi_{k-1,n}(p,q)$ holds on a word if p and q occur exactly once and, if the position at which p occurs is i, the position at which q occurs is $i+h(k-1,n)$. In addition to the use of these formulas, we use universally quantified propositions that mark potential violations of the conditions. To illustrate this idea, we sketch a formula that expresses that a run of $\mathcal{M}$ eventually contains a finite word starting with *start* and ending in *end* in which tiles are followed by exactly $h(k-1,n)$-many bits. The atomic proposition *tile* holds if the current letter encodes a tile.

$$\begin{aligned} &\forall d.\Big(\big[\forall p\forall q\big(\varphi_{k-1,n}(p,q)\to\Box[(d\wedge \textit{tile}\wedge p)\to\\ &\qquad \text{next occurrence of } \textit{tile} \text{ or } \textit{end} \text{ not one step after } q]\big)\big]\to\Diamond(\textit{start}\wedge(\neg(d\,\mathrm{U}\,\textit{end}))\Big). \end{aligned}$$

The quantified proposition d can be used to mark any tiles for which the next tile or *end* does not follow exactly $h(k-1,n)+1$ steps later. The quantified variables p and q and the formula $\varphi_{k-1,n}(p,q)$ are used to check that the markers are placed correctly, i.e., that indeed the next occurrence of *tile* or *end* after the marked position is not exactly $h(k-1,n)+1$ steps later. If the markers d are not placed correctly, the formula holds. Otherwise, it holds if a finite word between *start* and *end* is contained in the run in which no tile is marked by d. As d is universally quantified, the formula hence holds on a run of $\mathcal{M}$ iff it contains a finite word starting with *start* and ending in *end* in which tiles are followed by exactly $h(k-1,n)$-many bits. Note that $\varphi_{k-1,n}(p,q)$ occurs in the scope of two negations due to the implications while $\forall p \forall q$ occurs in the scope of one negation. So, the formula is in Π_k^{QLTL}.

The correctness of the counter can be expressed using the same idea of marking bits that violate the correctness of the counter with a universally quantified variable and the fact that a bit in a binary counter changes during an increment of the counter if and only if all less significant bits are 1. The vertical condition of the tiling is checked by using universally quantified markers ν_1 and ν_2 that have to be placed on vertically adjacent tiles. The correct placement of the markers is checked by stating that there exists a proposition b that encodes a correct binary counter with $h(k-1,n)$-many bits that starts with $0\dots0$ after ν_1 and counts up to $1\dots1$ right before ν_2. The correctness of the counter is checked as for the counter using the bits 0 and 1. The additional existential quantification over b does not yield an additional quantifier alternation. The resulting formula *valid_tiling* is in Π_k^{QLTL} and holds on a run of $\mathcal{M}$ if an encoding of a valid tiling is produced. As a run of $\mathcal{M}$ almost surely contains all words in $\mathit{start}(T\cup\{0,1\})^+\mathit{end}$, the formula *valid_tiling* holds with probability 1 iff it holds with positive probability iff there is a valid tiling for the given instance of the k-exponential tiling problem. ◀

As the upper bounds are obtained via the construction of NBAs for the QLTL-formulas, we can conclude the same results for the quantitative model-checking problems over Markov chains.

▶ **Corollary 3** (Quantitative model checking). *Given a Σ_k^{QLTL}- or Π_k^{QLTL}-formula φ and a Markov chain $\mathcal{M}$, the probability $\Pr_{\mathcal{M}}(\varphi)$ can be computed in k-exponential space and in time polynomial in the size of $\mathcal{M}$. Given a rational $\vartheta \in [0,1]$ and $\bowtie \in \{\leqslant,<,>,\geqslant\}$, deciding whether $\Pr_{\mathcal{M}}(\varphi)\bowtie\vartheta$ is k-EXPSPACE-complete.*

Proof. The lower bounds follow directly from the previous theorem. The upper bound follows from the fact that, given a Markov chain $\mathcal{M}$ and an NBA $\mathcal{A}$, the probability $\Pr_{\mathcal{M}}(\mathcal{A})$ that a word produced by $\mathcal{M}$ is accepted by $\mathcal{A}$ can be computed in time polynomial in $\mathcal{M}$ [7] and in space polynomial in the total size of the input. We sketch a proof of the latter claim: In the algorithm provided by Courcoubetis and Yannakakis in [7] to compute this probability, an exponentially large Markov chain $\mathcal{N}$ is constructed from $\mathcal{M}$ and $\mathcal{A}$. The states of $\mathcal{N}$ have a polynomial representation in the size of $\mathcal{M}$ and $\mathcal{A}$ and one can compute the transition probabilities between any two states in polynomial time. The probability $\Pr_{\mathcal{M}}(\mathcal{A})$ now equals the probability to reach a *recurrent* state in $\mathcal{N}$ – as it is called in [7], but which we do not define here. It is only important to us that one can decide whether a state is recurrent in polynomial space polynomial in the size of $\mathcal{A}$ (and polylogarithmic in the size of $\mathcal{M}$) as shown in [7]. The probability to reach a recurrent state in $\mathcal{N}$ can be computed by solving a linear equation system. As transition probabilities and whether states are recurrent in $\mathcal{N}$ can be computed in space polynomial in $\mathcal{A}$, each entry of the matrix and vector representing this linear equation system, which is of size exponential in $\mathcal{A}$ and polynomial in $\mathcal{M}$, can be computed in space polynomial in $\mathcal{A}$. Using the fact that solving linear equation systems lies in the complexity class NC and can hence be done in polylogarithmic space (see, e.g., [18, Section 15]) and standard results on the composition of space-bounded transductions (see, e.g., [18, Section 8]), we can conclude that the probability $\Pr_{\mathcal{M}}(\mathcal{A})$ can be computed in space polynomial in the size of $\mathcal{A}$. Applied to the k-exponentially sized NBAs for Σ_k^{QLTL}-formulas, this result leads to the claim of the corollary. ◀

3.2 Markov decision processes

We now provide the complexity results for QLTL-model checking over MDPs.

▶ **Theorem 4.** *Given an MDP $\mathcal{M}$, a Π_k^{QLTL}-formula φ, and* $\mathrm{opt} \in \{\max, \min\}$*, deciding whether* $\Pr_{\mathcal{M}}^{\mathrm{opt}}(\varphi) = 1$ *and deciding whether* $\Pr_{\mathcal{M}}^{\mathrm{opt}}(\varphi) > 0$ *are* $k+1$*-EXPTIME-complete for any* $k \geqslant 1$*. The problems are solvable in time polynomial in the size of $\mathcal{M}$.*

As Π_k^{QLTL} is not closed under negation, the model-checking problems in MDPs concerning the maximal and minimal satisfaction probability, respectively, require different hardness proofs. We sketch the two proof ideas in the sequel. The full proofs can be found in [19].

Proof sketch. The upper bounds are obtained via the straight-forward construction of *deterministic* automata (e.g., deterministic Rabin automata; see, e.g., [2]). This requires the determinization of the k-exponentially large NBAs for Σ_k^{QLTL}-formulas, which are computable in k-exponential time, and leads to a $k+1$-exponential-time procedure.

For the lower bounds, first consider the problems with $\mathrm{opt} = \max$. We prove $k+1$-EXPTIME-hardness by encoding the computation of k-exponentially space-bounded alternating Turing machines (ATM). It is well-known that the class of problems decidable by such ATMs coincides with $k+1$-EXPTIME [5]. So, given a k-exponentially space-bounded ATM $\mathcal{T}$ and an input word w, we construct an MDP $\mathcal{M}$ and a Π_k^{QLTL}-formula φ such that $\Pr_{\mathcal{M}}^{\max}(\varphi) > 0$ iff $\Pr_{\mathcal{M}}^{\max}(\varphi) = 1$ iff w is accepted by $\mathcal{T}$. Recall that acceptance in an ATM can be specified in terms of a game between a universal player choosing the next move in universal states and an existential player choosing the next move in existential state. A word is accepted if the existential player has a strategy that ensures that an accepting state is reached from the initial configuration with the input word on the tape.

The idea for the reduction is to construct an MDP $\mathcal{M}$ in which the scheduler can produce a sequence of (k-exponentially long) configurations of $\mathcal{T}$. The sequence of configurations in turn represents a sequence of infinitely many finite computations. The first configuration of each computation has to be the initial configuration with w on the tape. After each configuration, the scheduler has to specify whether the universal or existential player has to choose the next move, or whether the computation ended and a new computation is about to start. If it is the existential player's turn, the scheduler chooses a move and has to construct the successor configuration accordingly. If it is the universal player's turn, the successor move is specified by a random choice and again the scheduler has to construct the correct successor configuration. The constructed MDP is sketched in Figure 2.

The Π_k^{QLTL}-formula φ we construct, on the one hand, expresses that the sequence produced by the scheduler obeys all these requirements. Checking that the successor configurations are constructed correctly is possible with the use of the Σ_k^{QLTL}-formulas $\varphi_{k,n}(p,q)$ from [22] that express that the positions at which p and q are a fixed k-exponentially large number of steps apart. On the other hand, the formula φ expresses that all (infinitely many) encoded computations end in an accepting state. If w is accepted by $\mathcal{T}$, the scheduler can construct correct accepting computations no matter what moves are chosen by the universal player and so $\Pr_{\mathcal{M}}^{\max}(\varphi) = 1$. If w is not accepted, however, the universal player will play according to a winning strategy in any of the encoded computations with positive probability. So, almost surely at some point any scheduler has to violate one of the requirements or construct a rejecting computation. In this case, $\Pr_{\mathcal{M}}^{\max}(\varphi) = 0$.

In contrast to the case just discussed, the statement $\Pr_{\mathcal{M}}^{\min}(\varphi) = 1$ is a statement about all schedulers. So, we cannot let a scheduler construct sequences of computations anymore. Instead, we construct an MDP $\mathcal{M}'$ that randomly generates sequences that potentially encode

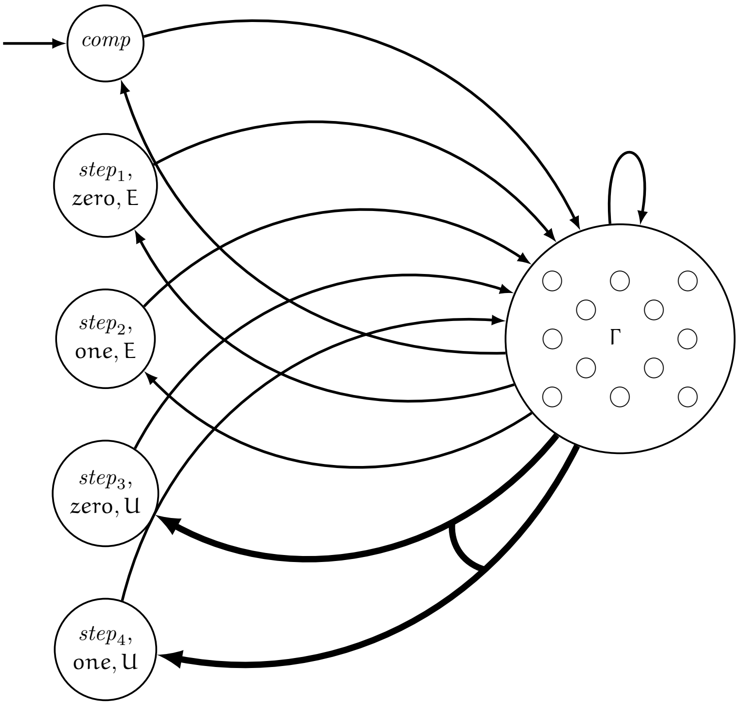


Figure 2 The MDP $\mathcal{M}$. The state depicted as Γ represents the behavior of each state $\gamma \in \Gamma$. I.e. from each state, there is one action to each state in Γ with probability 1. Further, from all states in Γ, there are actions leading to states *comp*, $step_1$ and $step_2$ with probability 1, as well as a randomized action (bold lines) leading to states $step_3$ and $step_4$ with probability 1/2 each. The labels zero and one indicate which successor move was chosen according to which the new configuration has to be constructed. The labels E and U indicate which player has chosen the move and are used to check whether the successor move was indeed chosen randomly iff it is the universal player's move.

correct computations. In the acceptance game of the given ATM, we also switch roles and let the choices of the existential player be made randomly while the scheduler can specify which successor move should be chosen in universal states. With positive probability, the correct successor configuration will be generated afterwards. Hence, if the existential player has a winning strategy, a correct accepting computation will eventually be produced randomly with probability 1 no matter what successor moves a scheduler chooses. Otherwise, there is a scheduler that prohibits this.

In order to express that eventually a correct accepting computation is generated in Π_k^{QLTL}, however, it turns out that we cannot use the Σ_k^{QLTL}-formulas $\varphi_{k,n}(p,q)$ from [22] as before. This is in part due to the implicit existential quantification in the eventually-modality. For this reason, we do not encode the computations simply as concatenations of configurations. Instead, we employ the ideas that were also used in the hardness proof for Markov chains (Theorem 2): We separate the symbols of the configurations by $k-1$-exponentially long binary counters to check that configurations have the correct length and use universally quantified variables to mark violations to any of the requirements of a valid encoding of an accepting computation. The blocks of the potential binary counter values are also randomly generated as sketched in Figure 3. An existentially quantified proposition encoding a further binary counter with $k-1$-exponentially many bits is then used to compare tape cells at the same position in two successive configurations, which are k-exponentially many steps apart

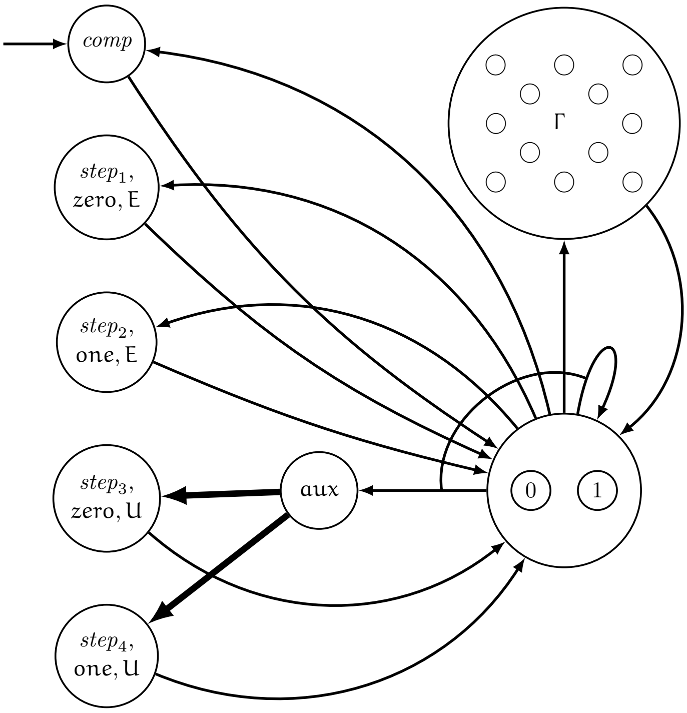


Figure 3 The MDP $\mathcal{M}'$. The behavior is probabilistic except for the choice in the state aux. When entering the cluster of states Γ or the cluster with the two bits 0 and 1, one of the states in the cluster is chosen randomly. Further all states in Γ have only the outgoing transition randomly moving to 0 or 1. The state aux is only an auxiliary state for the graphical representation. That means that in states 0 and 1 two actions are enabled. The first moving randomly to any state except for *step*$_3$; the second moving randomly to any state except for *step*$_4$.

in the encoding. Under any scheduler, the resulting Π_k^{QLTL}-formula φ holds on an execution of $\mathcal{M}$ almost surely if w is accepted by $\mathcal{T}$. Similar to before, each of the randomly generated potential computations is correct with positive probability and in each of these computations the randomly chosen moves of the existential player are in accordance with a winning strategy with positive probability against any scheduler, which chooses the moves of the universal player. If w is not accepted by $\mathcal{T}$, however, there is a strategy for the universal player and hence a scheduler that makes sure that no correct accepting computation is generated. In this case, $\Pr_{\mathcal{M}}^{\min}(\varphi) = 0$. ◀

These results allow us to conclude that all qualitative model-checking problems for Σ_k^{QLTL}-formulas in MDPs are $k+1$-EXPTIME-complete for any $k \geqslant 1$, too, as the negation of a Σ_k^{QLTL}-formula is a Π_k^{QLTL}-formula. Furthermore, as the upper bounds are obtained via the naive construction of deterministic automata, also the quantitative model checking problems have the same complexity as the minimal and maximal probabilities that an execution of an MDP is accepted by a suitable deterministic automaton (such as a deterministic Rabin automaton) can be computed in polynomial time (for details see, e.g., [2]).

▶ **Corollary 5** (Quantitative model checking). *Given a Σ_k^{QLTL}- or Π_k^{QLTL}-formula φ and an MDP $\mathcal{M}$, the probabilities $\Pr_{\mathcal{M}}^{\min}(\varphi)$ and $\Pr_{\mathcal{M}}^{\max}(\varphi)$ can be computed in time $k+1$-exponential in the size of φ and polynomial in the size of $\mathcal{M}$. Given a rational $\vartheta \in [0,1]$, $\bowtie \in \{\leqslant, <, >, \geqslant\}$ and $\mathrm{opt} \in \{\min, \max\}$, deciding whether $\Pr_{\mathcal{M}}^{\mathrm{opt}}(\varphi) \bowtie \vartheta$ is $k+1$-EXPSPACE-complete.*

4 Trace Vacuity in Probabilistic Systems

Vacuity notions have been studied for non-probabilistic systems in order to express, roughly, that the truth of a formula is not affected by the truth of one of its subformulae [1, 3, 16]. Among the existing definitions of vacuity in the literature, *trace vacuity* is the strongest.

▶ **Definition 6.** *Let φ be an LTL-formula and ψ a subformula. Let $\mathcal{T}$ be a transition system. We say that ψ* does not affect *φ in $\mathcal{T}$ if for every execution π in $\mathcal{T}$:*

$$\pi \models \forall x.\varphi[\psi \leftarrow x] \quad \Longleftrightarrow \quad \pi \models \exists x.\varphi[\psi \leftarrow x].$$

We say that φ holds vacuously in $\mathcal{T}$ if there is a subformula that does not affect φ in $\mathcal{T}$.

The above definition of non-affection generalizes the one from [1] by relaxing the hypothesis that φ holds on $\mathcal{T}$. For any execution π, $\pi \models \forall x.\varphi[\psi \leftarrow x] \Rightarrow \pi \models \varphi \Rightarrow \pi \models \exists x.\varphi[\psi \leftarrow x]$. We thus merely require that the three sets of executions that satisfy, $\forall x.\varphi[\psi \leftarrow x]$, φ, and $\exists x.\varphi[\psi \leftarrow x]$ respectively, coincide. Also, this generalisation allows us to naturally extend the notions of non-affection and vacuity to probabilistic systems. In the remainder of this section, we introduce trace vacuity for probabilistic systems, and establish tight complexity bounds for checking probabilistic vacuity. As in the non-probabilistic case, vacuity checking reduces to checking a Π_1^{QLTL}-formula. Conversely, one can reduce the qualitative model checking of Π_1^{QLTL} to probabilistic vacuity.

4.1 Probabilistic trace vacuity

▶ **Definition 7.** *Let φ be an LTL-formula and ψ a subformula. Let $\mathcal{M}$ be an MDP or a Markov chain. We say that ψ* does not affect *φ in $\mathcal{M}$ iff*

$$\mathrm{Pr}^{\min}_{\mathcal{M}}(\forall x.(\varphi[\psi \leftarrow x] \leftrightarrow \varphi)) = 1.$$

We say that φ is vacuous in $\mathcal{M}$ if there is a subformula that does not affect φ in $\mathcal{M}$.

Note that it does make sense for Markov chains and MDPs to consider that a formula is vacuous if its satisfaction probability (under any scheduler) is not affected when replacing a subformula, even if the global formula does not hold almost-surely. In MDPs, the definition of non-affection generalizes the non-probabilistic definition. This is made more precise in the following proposition. Paths in a transition system correspond to schedulers not making use of randomization when we view a transition system as an MDP.

▶ **Proposition 8.** *A subformula ψ does not affect a formula φ in an MDP (or a Markov chain) $\mathcal{M}$ if and only if for all schedulers $\mathfrak{S}$, $\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\forall x.\varphi[\psi \leftarrow x]) = \mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\varphi) = \mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}}(\exists x.\varphi[\psi \leftarrow x])$.*

Proof. Let us rewrite $\forall x.(\varphi[\psi \leftarrow x] \leftrightarrow \varphi)$ as $(\forall x.(\varphi \rightarrow \varphi[\psi \leftarrow x])) \wedge (\forall x.(\varphi[\psi \leftarrow x] \rightarrow \varphi))$. The latter is equivalent to $(\varphi \rightarrow \forall x.\varphi[\psi \leftarrow x]) \wedge (\forall x.\neg\varphi[\psi \leftarrow x] \vee \varphi)$. Rewritten as implications, we obtain $(\varphi \rightarrow \forall x.\varphi[\psi \leftarrow x]) \wedge (\exists x.\varphi[\psi \leftarrow x] \rightarrow \varphi)$. As the two implications $(\varphi \leftarrow \forall x.\varphi[\psi \leftarrow x])$ and $(\exists x.\varphi[\psi \leftarrow x] \leftarrow \varphi)$ are tautologies, the claim follows easily considering that the minimal probability in Definition 7 can be read as a universal quantification over schedulers. ◀

▶ **Example 9.** We provide a short example of non-affection in Markov chains, also to shed light on the difference with the non-probabilistic setting. Consider the Markov chain on Fig. 4, where we assume arbitrary non-zero probabilities on edges, and the following

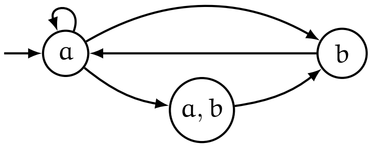


Figure 4 A Markov chain to illustrate the notion of affection.

formulas: $\varphi = \Box\Diamond(a \wedge b) \vee \Box(a \vee b)$ and $\psi = \Box(a \vee b)$. Clearly enough, $\Pr_{\mathcal{M}}(\varphi) = \Pr_{\mathcal{M}}(\exists x.\varphi[\psi \leftarrow x]) = \Pr_{\mathcal{M}}(\forall x.\varphi[\psi \leftarrow x]) = 1$ so that ψ does not affect φ, and φ is vacuous in this Markov chain. However, if one views the graph as a transition system $\mathcal{T}$, then $\mathcal{T} \models \varphi$ and $\mathcal{T} \not\models \forall x.\varphi[\psi \leftarrow x]$. So, ψ affects φ.

Armoni *et al.* [1] observed that if ψ appears only positively in φ, for every execution π in the transition system $\mathcal{T}$ then: $\mathcal{T}, \pi \models \forall x.\varphi[\psi \leftarrow x] \iff \mathcal{T}, \pi \models \varphi[\psi \leftarrow \bot]$. As a consequence, a pure polarity subformulas ψ does not affect φ if and only if $\Pr^{\min}_{\mathcal{M}}(\varphi[\psi \leftarrow \top] \leftrightarrow \varphi[\psi \leftarrow \bot]) = 1$. Therefore, checking whether a pure polarity subformula affects a formula reduces to quantitative model checking of LTL formulas and can be done in PSPACE for Markov chains and in 2-EXPTIME for MDPs.

As also argued in [1], restricting attention to subformulas with pure polarity or to consider single occurrences of subformulas separately is insufficient for a satisfactory vacuity check. For example, a formula like $\Box(p \to p) \equiv \Box(p \vee \neg p)$, in which p occurs positively and negatively, should be rendered vacuous in any system. Restricting attention to only one of the two occurences of p, however, would in general lead to the insight that each of the two occurrences on its own does affect the formula. Beyond pure polarity formulas, checking affectation is harder for Markov chains. Indeed, hardness of Π_1^{QLTL} model checking transfers to hardness of vacuity checking. As stated in the next theorem, for MDPs affection checking has the same complexity as quantitative LTL model checking, whereas Markov chains exhibit an exponential complexity blowup.

▶ **Theorem 10.** *Checking whether a subformula ψ affects an LTL-formula φ in a Markov chain $\mathcal{M}$ is EXPSPACE-complete. In MDPs, the problem is 2-EXPTIME-complete.*

Proof. The upper bounds follow directly from the upper bounds of qualitative model-checking of Π_1^{QLTL} in Markov chains and MDPs. For the lower bound, we first concentrate on MDPs. We provide a reduction from the problem whether a Π_1^{QLTL}-formula $\vartheta = \forall x.\varphi$ satisfies $\Pr^{\min}_{\mathcal{M}}(\vartheta) = 1$ in an MDP $\mathcal{M}$. A proof that the restriction to one quantified variable does not influence the complexity is given in [19]. So, let $\mathcal{M}$ be labeled with atomic propositions from AP. Let $\vartheta = \forall x.\varphi$ where φ is an LTL-formula over $\mathsf{AP} \cup \{x\}$ with $x \notin \mathsf{AP}$ be given. We construct the MDP $\mathcal{M}'$ by adding a new initial state s'_{init} from which the original initial state s_{init} is reached in one step with probability 1. Further, we let β be an LTL-formula that is valid and does not occur in φ. Finally, we define φ' to be the LTL-formula $\varphi' = \beta \vee \bigcirc\varphi[x \leftarrow \beta]$. Of course, $\Pr^{\min}_{\mathcal{M}', s'_{init}}(\varphi') = 1$ as β is valid. We claim that β does not affect φ' in $\mathcal{M}'$ if and only if $\Pr^{\min}_{\mathcal{M}}(\forall x.\varphi) = 1$. The subformula β does not affect φ' in $\mathcal{M}'$ iff $\Pr^{\min}_{\mathcal{M}', s'_{init}}(\forall x.(x \vee \bigcirc\varphi)) = 1$ by definition and the fact that β does not occur anywhere else in φ. But $\Pr^{\min}_{\mathcal{M}', s'_{init}}(\forall x.(x \vee \bigcirc\varphi)) = 1$ holds if and only if $\Pr^{\min}_{\mathcal{M}', s'_{init}}(\forall x. \bigcirc \varphi) = 1$ because the universal quantifier can choose x not to hold in the first position of any trace produced by $\mathcal{M}'$. After the first step $\mathcal{M}'$ behaves exactly like $\mathcal{M}$ and hence $\Pr^{\min}_{\mathcal{M}', s'_{init}}(\forall x. \bigcirc \varphi) = 1$ if and only if $\Pr^{\min}_{\mathcal{M}, s_{init}}(\forall x.\varphi) = 1$. So, checking affection in MDPs is as hard as the respective qualitative model-checking problem for Π_1^{QLTL} and hence 2-EXPTIME-complete.

For Markov chains, the argument goes analogously. Note that the constructed MDP $\mathcal{M}'$ is a Markov chain if $\mathcal{M}$ is a Markov chain. So, checking affection in Markov chains is also as hard as the respective qualitative model-checking problem for Π_1^{QLTL} and hence EXPSPACE-complete. ◀

In Markov chains, the exponential blow-up in complexity of non-affection checking compared to LTL-model checking constitutes a major obstacle for vacuity checking. To provide a possibility to check that a specification is not obviously faulty without such an exponential blow-up, we turn our attention to the notion of inherent vacuity.

4.2 Inherent vacuity in probabilistic systems

Inherent vacuity for transition systems expresses whether a formula holds vacuously in every model in which it holds [9]. Using our generalized definition, we do not restrict ourselves to the models in which the formula holds anymore and provide an analogous definition for probabilistic systems. As in [9], we consider two natural variants of the definition and investigate how to check whether a formula is inherently vacuous.

▶ **Definition 11.** *Let φ be an LTL-formula. Let $\mathcal{C}$ be the class of all transition systems, all Markov chains, or all MDPs, respectively. We say that φ is inherently vacuous over $\mathcal{C}$, if φ is vacuous in all models $\mathcal{M} \in \mathcal{C}$. For a subformula ψ of φ we say that ψ inherently does not affect φ over $\mathcal{C}$, if for every $\mathcal{M} \in \mathcal{C}$, ψ does not affect φ in $\mathcal{M}$. If there is a subformula that inherently does not affect φ over $\mathcal{C}$, we say that φ is uniformly inherently vacuous.*

In [9], it is shown that inherent vacuity and uniform inherent vacuity coincide for transition systems. Dropping the restriction to models in which a formula φ holds, the results of [9] show that the notions are equivalent to the existence of a subformula ψ such that $\forall x.(\varphi[\psi \leftarrow x] \leftrightarrow \varphi)$ is valid. We prove that inherent and uniform inherent vacuity for Markov chains and MDPs are also equivalent to this condition and hence to inherent vacuity in transition systems. First, we show that uniform inherent vacuity coincides with inherent vacuity.

▶ **Proposition 12.** *Let φ be an LTL-formula and let $\mathcal{C}$ be the class of all Markov chains or all MDPs. The formula φ is uniformly inherently vacuous over $\mathcal{C}$ if and only if it is inherently vacuous over $\mathcal{C}$.*

Proof. One direction is clear. For the other direction, suppose that φ is inherently vacuous over $\mathcal{C}$, but not uniformly inherently vacuous. Hence, for each subformula ψ of φ, there is a model $\mathcal{M}_\psi \in \mathcal{C}$ such that ψ affects φ over $\mathcal{M}_\psi$. Let $\mathcal{N}$ be the disjoint union of the models $\mathcal{M}_\psi$ for all subformulas ψ with an initial uniform probability distribution over the initial states of these models. We claim that φ is not vacuous in $\mathcal{M}$. For each subformula ψ, there is a positive probability that $\mathcal{M}_\psi$ is chosen. As there is a scheduler $\mathfrak{S}$ (for Markov chains, the unique scheduler) with $\mathrm{Pr}^{\mathfrak{S}}_{\mathcal{M}_\psi}(\forall x.(\varphi[\psi \leftarrow x] \leftrightarrow \varphi) < 1$, the same holds in $\mathcal{N}$. This is a contradiction to the inherent vacuity of φ. ◀

The following proposition establishes that all variants of inherent vacuity considered coincide:

▶ **Proposition 13.** *Let φ be an LTL-formula and ψ a subformula. Then, ψ inherently does not affect φ over Markov chains or MDPs, respectively, if and only if the formula $\forall x.(\varphi \leftrightarrow \varphi[\psi \leftarrow x])$ is valid.*

Proof. Only the left-to-right implication deserves a proof, and we prove the contrapositive. Assume the formula $\chi = \forall x.(\varphi \leftrightarrow \varphi[\psi \leftarrow x])$ is not valid. Since χ expresses a regular property, there exists an ultimately periodic word w that violates χ. It suffices to consider the Markov chain or MDP $\mathcal{M}$ that has only one path, and produces w with probability 1, and observe that ψ does affect φ in $\mathcal{M}$. ◀

As a consequence, checking inherent vacuity for probabilistic systems is as simple as in the non-probabilistic case, and can be done in polynomial space. In particular for Markov chains, an inherent vacuity check might be an interesting option for practical applications as it avoids the exponential blow-up in complexity over LTL-model checking.

5 Conclusion

We determined the precise complexities of the model-checking problems for the different levels of the quantifier alternation hierarchy of QLTL over probabilistic systems. The knowledge of the precise complexities, in particular the established lower bounds, has the potential to serve as the basis for hardness proofs for other questions in the formal verification of probabilistic systems. Despite the high complexities that we obtained, efficient model checking for formulas with few quantifier alternations might still be possible because all problems are solvable in time polynomial in the size of the system and typically formulas are small compared to the size of the models.

These results have been applied to the notion of trace vacuity known from the non-probabilistic setting that we adapted to the probabilistic setting. It turned out that checking whether a formula is affected by a subformula in a system is inter-reducible with Π_1^{QLTL}-model checking. For Markov chains, our new lower bounds allowed us to conclude that affection checking is EXPSPACE-complete and hence exponentially harder than LTL-model checking, while the complexity of affection checking and LTL-model checking are the same in MDPs. Furthermore, we showed that the notion of inherent vacuity – expressing that a formula is vacuous in a class of system models – is invariant under the switch from non-probabilistic to probabilistic models, and hence, known polynomial-space algorithms are applicable for Markov chains and MDPs. In addition to the vacuity notions we studied here, an interesting direction for future research is the investigation of “more probabilistic” notions of vacuity that express that a perturbation of a subformula does not influence the satisfaction probability of a formula in a system.

References

1 Roy Armoni, Limor Fix, Alon Flaisher, Orna Grumberg, Nir Piterman, Andreas Tiemeyer, and Moshe Y. Vardi. Enhanced vacuity detection in linear temporal logic. In *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 368–380. Springer, 2003.

2 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* MIT Press, 2008.

3 Ilan Beer, Shoham Ben-David, Cindy Eisner, and Yoav Rodeh. Efficient detection of vacuity in temporal model checking. *Formal Methods in System Design*, 18(2):141–163, 2001.

4 Doron Bustan, Alon Flaisher, Orna Grumberg, Orna Kupferman, and Moshe Y. Vardi. Regular vacuity. In *Proceedings of the 13th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'05)*, volume 3725 of *Lecture Notes in Computer Science*, pages 191–206. Springer, 2005.

5 Ashok K. Chandra, Dexter C. Kozen, and Larry J. Stockmeyer. Alternation. *Journal of the ACM*, 28(1):114–133, 1981. `doi:10.1145/322234.322243`.

6 Edmund Clarke, Orna Grumberg, and Doron Peled. *Model Checking.* MIT Press, 2000.

7 Costas Courcoubetis and Mihalis Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.

8 Bernd Finkbeiner and Leander Tentrup. Detecting unrealizability of distributed fault-tolerant systems. *Logical Methods in Computer Science*, 11(3):1–31, 2015. doi:10.2168/LMCS-11(3:12)2015.

9 Dana Fisman, Orna Kupferman, Sarai Sheinvald-Faragy, and Moshe Y. Vardi. A framework for inherent vacuity. In *Proceedings of the 4th International Haifa Verification Conference (HVC'08)*, volume 5394 of *Lecture Notes in Computer Science*, pages 7–22. Springer, 2008. doi:10.1007/978-3-642-01702-5_7.

10 Dov Gabbay. The declarative past and imperative future. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Temporal Logic in Specification*, pages 409–448, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg.

11 Arie Gurfinkel and Marsha Chechik. Extending extended vacuity. In *Proceedings of the 5th International Conference on Formal Methods in Computer-Aided Design (FMCAD'04)*, volume 3312 of *Lecture Notes in Computer Science*, pages 306–321. Springer, 2004.

12 Arie Gurfinkel and Marsha Chechik. How vacuous is vacuous? In *Proceedings of the 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, pages 451–466, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

13 Walter Hussak. Serializable histories in quantified propositional temporal logic. *International Journal of Computer Mathematics*, 81(10):1203–1211, 2004. doi:10.1080/00207160412331284051.

14 Yonit Kesten and Amir Pnueli. Complete proof system for QPTL. *Journal of Logic and Computation*, 12(5):701–745, 2002.

15 Orna Kupferman. Sanity checks in formal verification. In *Proceedings of the 17th International Conference on Concurrency Theory (CONCUR'06)*, volume 4137 of *Lecture Notes in Computer Science*, pages 37–51. Springer, 2006.

16 Orna Kupferman and Moshe Y. Vardi. Vacuity detection in temporal model checking. *International Journal on Software Tools for Technology Transfer*, 4(2):224–233, 2003.

17 François Laroussinie and Nicolas Markey. Quantified CTL: Expressiveness and Complexity. *Logical Methods in Computer Science*, 10(4):1–45, 2014. doi:10.2168/LMCS-10(4:17)2014.

18 Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

19 Jakob Piribauer, Christel Baier, Nathalie Bertrand, and Ocan Sankur. Quantified linear temporal logic over probabilistic systems with an application to vacuity checking (extended version). Technical report, TU Dresden, Dresden, Germany, 2021. See https://wwwtcs.inf.tu-dresden.de/ALGI/PUB/CONCUR21/.

20 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, 1994.

21 A. Prasad Sistla. *Theoretical Issues in the Design and Verification of Distributed Systems.* PhD thesis, Carnegie-Mellon University, 1983.

22 A. Prasad Sistla, Moshe Y. Vardi, and Pierre Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49(2–3):217–237, 1987.

23 Aravinda P. Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985. doi:10.1145/3828.3837.

24 Peter van Emde Boas. The convenience of tilings. *Lecture Notes in Pure and Applied Mathematics*, pages 331–363, 1997.

25 Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *Proceedings of the 1st Symposium on Logic in Computer Science (LICS'86)*, pages 332–344. IEEE Computer Society Press, 1986.

26 Pierre Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1):72–99, 1983. doi:10.1016/S0019-9958(83)80051-5.

Subgame-Perfect Equilibria in Mean-Payoff Games

Léonard Brice ✉
LIGM, Univ. Gustave Eiffel, CNRS, F-77454 Marne-la-Vallée, France

Jean-François Raskin ✉
Université libre de Bruxelles, Brussels, Belgium

Marie van den Bogaard ✉
LIGM, Univ. Gustave Eiffel, CNRS, F-77454 Marne-la-Vallée, France

Abstract

In this paper, we provide an effective characterization of all the subgame-perfect equilibria in infinite duration games played on finite graphs with mean-payoff objectives. To this end, we introduce the notion of requirement, and the notion of negotiation function. We establish that the plays that are supported by SPEs are exactly those that are consistent with the least fixed point of the negotiation function. Finally, we show that the negotiation function is piecewise linear, and can be analyzed using the linear algebraic tool box. As a corollary, we prove the decidability of the SPE constrained existence problem, whose status was left open in the literature.

2012 ACM Subject Classification Software and its engineering → Formal methods; Theory of computation → Logic and verification; Theory of computation → Solution concepts in game theory

Keywords and phrases Games on graphs, subgame-perfect equilibria, mean-payoff objectives.

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.8

Related Version *Full Version*: `https://arxiv.org/abs/2101.10685` [3]

Funding This work is partially supported by the ARC project Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond (Fédération Wallonie-Bruxelles), the EOS project Verifying Learning Artificial Intelligence Systems (F.R.S.-FNRS & FWO), the COST Action 16228 GAMENET (European Cooperation in Science and Technology), and by the PDR project *Subgame perfection in graph games* (F.R.S.- FNRS).

1 Introduction

The notion of Nash equilibrium (NE) is one of the most important and most studied solution concepts in game theory. A profile of strategies is an NE when no rational player has an incentive to change their strategy unilaterally, i.e. while the other players keep their strategies. Thus an NE models a stable situation. Unfortunately, it is well known that, in sequential games, NEs suffer from the problem of *non-credible threats*, see e.g. [18]. In those games, some NE only exists when some players do *not* play rationally in subgames and so use non-credible threats to force the NE. This is why, in sequential games, the stronger notion of *subgame-perfect equilibrium* is used instead: a profile of strategies is a subgame-perfect equilibrium (SPE) if it is an NE in all the subgames of the sequential game. Thus SPE imposes rationality even after a deviation has occured.

In this paper, we study sequential games that are infinite-duration games played on graphs with mean-payoff objectives, and focus on SPEs. While NEs are guaranteed to exist in infinite duration games played on graphs with mean-payoff objectives, it is known that it is not the case for SPEs, see e.g. [19, 5]. We provide in this paper a constructive characterization of the entire set of SPEs, which allows us to decide, among others, the SPE (constrained) existence problem. This problem was left open in previous contributions on the subject. More precisely, our contributions are described in the next paragraphs.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 8; pp. 8:1–8:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Contributions. First, we introduce two important new notions that allow us to capture NEs, and more importantly SPEs, in infinite duration games played on graphs with mean-payoff objectives[1]: the notion of *requirement* and the notion of *negotiation function*.

A requirement λ is a function that assigns to each vertex $v \in V$ of a game graph a value in $\mathbb{R} \cup \{-\infty, +\infty\}$. The value $\lambda(v)$ represents a requirement on any play $\rho = \rho_0\rho_1 \dots \rho_n \dots$ that traverses this vertex: if we want the player who controls the vertex v to follow ρ and to give up deviating from ρ, then the play must offer a payoff to this player that is at least $\lambda(v)$. An infinite play ρ is λ-consistent if, for each player i, the payoff of ρ for player i is larger than or equal to the largest value of λ on vertices occurring along ρ and controlled by player i.

We first use those notions to rephrase a classical result about NEs: if λ maps a vertex v to the largest value that the player that controls v can secure against a fully adversarial coalition of the other players, i.e. if $\lambda(v)$ is the zero-sum worst-case value, then the set of plays that are λ-consistent is exactly the set of plays that are supported by an NE (Theorem 24).

As SPEs are forcing players to play rationally in all subgames, we cannot rely on the zero-sum worst-case value to characterize them. Indeed, when considering the worst-case value, we allow adversaries to play fully adversarially after a deviation and so potentially in an irrational way w.r.t. their own objective. In fact, in an SPE, a player is refrained to deviate when opposed by a coalition of *rational adversaries*. To characterize this relaxation of the notion of worst-case value, we rely on our notion of *negotiation function*.

The negotiation function nego operates from the set of requirements into itself. To understand the purpose of the negotiation function, let us consider its application on the requirement λ that maps every vertex v on the worst-case value as above. Now, we can naturally formulate the following question: given v and λ, can the player who controls v improve the value that they can ensure against all the other players, if only plays that are consistent with λ are proposed by the other players? In other words, can this player enforce a better value when playing against the other players if those players are not willing to give away their own worst-case value? Clearly, securing this worst-case value can be seen as a minimal goal for any *rational* adversary. So $\mathrm{nego}(\lambda)(v)$ returns this value; and this reasoning can be iterated. One of the contributions of this paper is to show that the least fixed point λ^* of the negotiation function is exactly characterizing the set of plays supported by SPEs (Theorem 28).

To turn this fixed point characterization of SPEs into algorithms, we additionally draw links between the negotiation function and two classes of zero-sum games, that are called *abstract* and *concrete* negotiation games (see Theorem 32). We show that the latter can be solved effectively and allow, given λ, to compute $\mathrm{nego}(\lambda)$ (Lemma 36). While solving concrete negotiation games allows us to compute $\mathrm{nego}(\lambda)$ for any requirement λ, and even if the function $\mathrm{nego}(\cdot)$ is monotone and Scott-continuous, a direct application of the Kleene-Tarski fixed point theorem is not sufficient to obtain an effective algorithm to compute λ^*. Indeed, we give examples that require a transfinite number of iterations to converge to the least fixed point. To provide an algorithm to compute λ^*, we show that the function $\mathrm{nego}(\cdot)$ is piecewise linear and we provide an effective representation of this function (Theorem 41). This effective representation can then be used to extract all its fixed points and in particular its least fixed point using linear algebraic techniques, hence the decidability of the SPE (constrained) existence problem (Theorem 45). Finally, all our results are also shown to extend to ε-SPEs, those are quantitative relaxations of SPEs.

[1] A large part of our results apply to the larger class of games with prefix-independent objectives. For the sake of readability of this introduction, we focus here on mean-payoff games but the technical results in the paper are usually covering broader classes of games.

Related works. Non-zero sum infinite duration games have attracted a large attention in recent years, with applications targeting reactive synthesis problems. We refer the interested reader to the following survey papers [2, 7] and their references for the relevant literature. We detail below contributions more closely related to the work presented here.

In [6], Brihaye et al. offer a characterization of NEs in quantitative games for cost-prefix-linear reward functions based on the worst-case value. The mean-payoff is cost-prefix-linear. In their paper, the authors do not consider the stronger notion of SPE, which is the central solution concept studied in our paper. In [8], Bruyère et al. study secure equilibria that are a refinement of NEs. Secure equilibria are not subgame-perfect and are, as classical NEs, subject to non-credible threats in sequential games.

In [20], Ummels proves that there always exists an SPE in games with ω-regular objectives and defines algorithms based on tree automata to decide constrained SPE problems. Strategy logics, see e.g. [12], can be used to encode the concept of SPE in the case of ω-regular objectives with application to the rational synthesis problem [15] for instance. In [13], Flesch et al. show that the existence of ε-SPEs is guaranteed when the reward function is *lower-semicontinuous*. The mean-payoff reward function is neither ω-regular, nor lower-semicontinuous, and so the techniques defined in the papers cited above cannot be used in our setting. Furthermore, as already recalled above, see e.g. [21, 5], contrary to the ω-regular case, SPEs in games with mean-payoff objectives may fail to exist.

In [5], Brihaye et al. introduce and study the notion of weak subgame-perfect equilibria, which is a weakening of the classical notion of SPE. This weakening is equivalent to the original SPE concept on reward functions that are *continuous*. This is the case for example for the quantitative reachability reward function, on which Brihaye et al. solve the problem of the constrained existence of SPEs in [4]. On the contrary, the mean-payoff cost function is not continuous and the techniques used in [5], and generalized in [10], cannot be used to characterize SPEs for the mean-payoff reward function.

In [17], Meunier develops a method based on Prover-Challenger games to solve the problem of the existence of SPEs on games with a finite number of possible outcomes. This method is not applicable to the mean-payoff reward function, as the number of outcomes in this case is uncountably infinite.

In [14], Flesch and Predtetchinski present another characterization of SPEs on games with finitely many possible outcomes, based on a game structure that we will present here under the name of *abstract negotiation game*. Our contributions differ from this paper in two fundamental aspects. First, it lifts the restriction to finitely many possible outcomes. This is crucial as mean-payoff games violate this restriction. Instead, we identify a class of games, that we call *with steady negotiation*, that encompasses mean-payoff games and for which some of the conceptual tools introduced in that paper can be generalized. Second, the procedure developed by Flesch and Predtetchinski is *not* an algorithm in CS acceptation: it needs to solve infinitely many games that are not represented effectively, and furthermore it needs a transfinite number of iterations. On the contrary, our procedure is effective and leads to a complete algorithm in the classical sense: with guarantee of termination in finite time and applied on effective representations of games.

Structure of the paper. In Sect. 2, we introduce the necessary background. Sect. 3 defines the notion of requirement and the negotiation function. Sect. 4 shows that the set of plays that are supported by an SPE are those that are λ^*-consistent, where λ^* is the least fixed point of the negotiation function. Sect. 5 draws a link between the negotiation function and negotiation games. Sect. 6 establishes that the negotiation function is effectively piecewise linear. Finally, Sect. 7 applies those results to prove the decidability of the SPE constrained existence problem on mean-payoff games, and adds some complexity considerations. The detailed proofs of our results, as well as additional examples, can be found in appendices of [3], the full version of this paper.

2 Background

In all what follows, we will use the word *game* for the infinite duration turn-based quantitative games on finite graphs with complete information.

▶ **Definition 1** (Game). A *game* is a tuple $G = (\Pi, V, (V_i)_{i \in \Pi}, E, \mu)$, where:

- Π is a finite set of *players*;
- (V, E) is a finite directed graph, whose vertices are sometimes called *states* and whose edges are sometimes called *transitions*, and in which every state has at least one outgoing transition. For the simplicity of writing, a transition $(v, w) \in E$ will often be written vw.
- $(V_i)_{i \in \Pi}$ is a partition of V, in which V_i is the set of states *controlled* by player i;
- $\mu : V^\omega \to \mathbb{R}^\Pi$ is an *outcome function*, that maps each infinite word ρ to the tuple $\mu(\rho) = (\mu_i(\rho))_{i \in \Pi}$ of the players' *payoffs*.

▶ **Definition 2** (Initialized game). An *initialized game* is a tuple (G, v_0), often written $G_{\upharpoonright v_0}$, where G is a game and $v_0 \in V$ is a state called *initial state*. Moreover, the game $G_{\upharpoonright v_0}$ is *well-initialized* if any state of G is accessible from v_0 in the graph (V, E).

▶ **Definition 3** (Play, history). A *play* (resp. history) in the game G is an infinite (resp. finite) path in the graph (V, E). It is also a play (resp. history) in the initialized game $G_{\upharpoonright v_0}$, where v_0 is its first vertex. The set of plays (resp. histories) in the game G (resp. the initialized game $G_{\upharpoonright v_0}$) is denoted by $\mathrm{Plays}G$ (resp. $\mathrm{Plays}G_{\upharpoonright v_0}, \mathrm{Hist}G, \mathrm{Hist}G_{\upharpoonright v_0}$). We write $\mathrm{Hist}_i G$ (resp. $\mathrm{Hist}_i G_{\upharpoonright v_0}$) for the set of histories in G (resp. $G_{\upharpoonright v_0}$) of the form hv, where v is a vertex controlled by player i.

▶ Remark. In the literature, the word *outcome* can be used to name plays, and the word *payoff* to name what we call here outcome. Here, the word *payoff* will be used to refer to outcomes, seen from the point of view of a given player – or in other words, an *outcome* will be seen as the collection of all players' payoffs.

▶ **Definition 4** (Strategy, strategy profile). A *strategy* for player i in the initialized game $G_{\upharpoonright v_0}$ is a function $\sigma_i : \mathrm{Hist}_i G_{\upharpoonright v_0} \to V$, such that $v\sigma_i(hv)$ is an edge of (V, E) for every hv. A history h is *compatible* with a strategy σ_i if and only if $h_{k+1} = \sigma_i(h_0 \dots h_k)$ for all k such that $h_k \in V_i$. A play ρ is compatible with σ_i if all its prefixes are.

A *strategy profile* for $P \subseteq \Pi$ is a tuple $\bar{\sigma}_P = (\sigma_i)_{i \in P}$, where for each i, σ_i is a strategy for player i in $G_{\upharpoonright v_0}$. A *complete* strategy profile, usually written $\bar{\sigma}$, is a strategy profile for Π. A play or a history is *compatible* with $\bar{\sigma}_P$ if it is compatible with every σ_i for $i \in P$.

When i is a player and when the context is clear, we will often write $-i$ for the set $\Pi \setminus \{i\}$. We will often refer to $\Pi \setminus \{i\}$ as the *environment* against player i. When $\bar{\tau}_P$ and $\bar{\tau}'_Q$ are two strategy profiles with $P \cap Q = \emptyset$, $(\bar{\tau}_P, \bar{\tau}'_Q)$ denotes the strategy profile $\bar{\sigma}_{P \cup Q}$ such that $\sigma_i = \tau_i$ for $i \in P$, and $\sigma_i = \tau'_i$ for $i \in Q$.

Before moving on to SPEs, let us recall the notion of Nash equilibrium.

▶ **Definition 5** (Nash equilibrium). Let $G_{\upharpoonright v_0}$ be an initialized game. The strategy profile $\bar{\sigma}$ is a *Nash equilibrium* – or *NE* for short – in $G_{\upharpoonright v_0}$ if and only if for each player i and for every strategy σ'_i, called *deviation of* σ_i, we have the inequality $\mu_i\left(\langle \sigma'_i, \bar{\sigma}_{-i} \rangle_{v_0}\right) \leq \mu_i\left(\langle \bar{\sigma} \rangle_{v_0}\right)$.

To define SPEs, we need the notion of subgame.

▶ **Definition 6** (Subgame, substrategy). Let hv be a history in the game G. The *subgame* of G after hv is the initialized game $(\Pi, V, (V_i)_i, E, \mu_{\upharpoonright hv})_{\upharpoonright v}$, where $\mu_{\upharpoonright hv}$ maps each play to its payoff in G, assuming that the history hv has already been played: formally, for every $\rho \in \mathrm{Plays}G_{\upharpoonright hv}$, we have $\mu_{\upharpoonright hv}(\rho) = \mu(h\rho)$.

If σ_i is a strategy in $G_{\upharpoonright v_0}$, its *substrategy* after hv is the strategy $\sigma_{i\upharpoonright hv}$ in $G_{\upharpoonright hv}$, defined by $\sigma_{i\upharpoonright hv}(h') = \sigma_i(hh')$ for every $h' \in \mathrm{Hist}_i G_{\upharpoonright hv}$.

▶ Remark. The initialized game $G_{\upharpoonright v_0}$ is also the subgame of G after the one-state history v_0.

▶ **Definition 7** (Subgame-perfect equilibrium). Let $G_{\upharpoonright v_0}$ be an initialized game. The strategy profile $\bar{\sigma}$ is a *subgame-perfect equilibrium* – or *SPE* for short – in $G_{\upharpoonright v_0}$ if and only if for every history h in $G_{\upharpoonright v_0}$, the strategy profile $\bar{\sigma}_{\upharpoonright h}$ is a Nash equilibrium in the subgame $G_{\upharpoonright h}$.

The notion of subgame-perfect equilibrium can be seen as a refinement of Nash equilibrium: it is a stronger equilibrium which excludes players resorting to non-credible threats.

▶ Example 8. In the game represented in Figure 1a, where the square state is controlled by player □ and the round states by player ○, if both players get the payoff 1 by reaching the state d and the payoff 0 in the other cases, there are actually two NEs: one, in blue, where □ goes to the state b and then player ○ goes to d, and both win, and one, in red, where player □ goes to the state c because player ○ was planning to go to e. However, only the blue one is an SPE, as moving from b to e is irrational for player ○ in the subgame $G_{\upharpoonright ab}$.

An ε-SPE is a strategy profile which is *almost* an SPE: if a player deviates after some history, they will not be able to improve their payoff by more than a quantity $\varepsilon \geq 0$.

▶ **Definition 9** (ε-SPE). Let $G_{\upharpoonright v_0}$ be an initialized game, and $\varepsilon \geq 0$. A strategy profile $\bar{\sigma}$ from v_0 is an ε-SPE if and only if for every history hv, for every player i and every strategy σ'_i, we have $\mu_i(\langle \bar{\sigma}_{-i\upharpoonright hv}, \sigma'_{i\upharpoonright hv} \rangle_v) \leq \mu_i(\langle \bar{\sigma}_{\upharpoonright hv} \rangle_v) + \varepsilon$.

Note that a 0-SPE is an SPE, and conversely.

Hereafter, we focus on *prefix-independent* games, and in particular *mean-payoff* games.

▶ **Definition 10** (Mean-payoff game). A *mean-payoff game* is a game $G = (\Pi, V, (V_i)_i, E, \mu)$, where μ is defined from a function $\pi : E \to \mathbb{Q}^\Pi$, called *weight function*, by, for each player i:

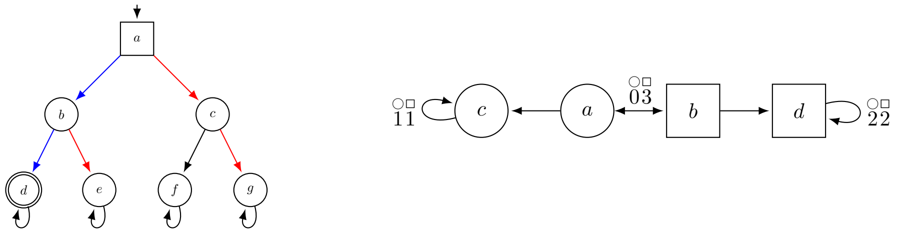
$$\mu_i : \rho \mapsto \liminf_{n \to \infty} \frac{1}{n} \sum_{k=0}^{n-1} \pi_i(\rho_k \rho_{k+1}).$$

In a mean-payoff game, the weight given by the function π represents the immediate reward that each action gives to each player. The final payoff of each player is their average payoff along the play, classically defined as the limit inferior over n (since the limit may not be defined) of the average payoff after n steps.

▶ **Definition 11** (Prefix-independent game). A game G is *prefix-independent* if, for every history h and for every play ρ, we have $\mu(h\rho) = \mu(\rho)$. We also say, in that case, that the outcome function μ is prefix-independent.

Mean-payoff games are prefix-independent. We now recall a classical result about two-player zero-sum games.

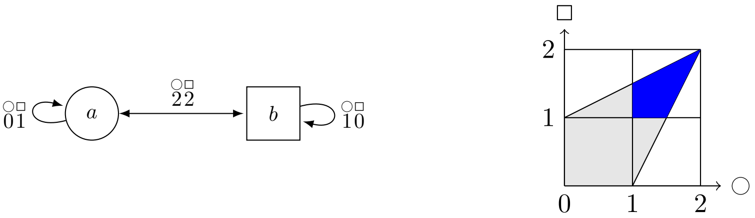
▶ **Definition 12** (Zero-sum game). A game G, with $\Pi = \{1, 2\}$, is *zero-sum* if $\mu_2 = -\mu_1$.



(a) Two NEs and one SPE.

(b) A game without SPE.

Figure 1 Two examples of games.



(a) The game G.

(b) The outcomes of plays and SPE plays in G.

Figure 2 A game with an infinity of SPEs.

▶ **Definition 13** (Borel game)**.** A game G is *Borel* if the function μ, from the set V^ω equipped with the product topology to the Euclidian space $\mathbb{R}^\Pi$, is Borel, i.e. if, for every Borel set $B \subseteq \mathbb{R}^\Pi$, the set $\mu^{-1}(B)$ is Borel.

▶ **Proposition 14** (Determinacy of two-player zero-sum Borel games, [16])**.** *Let $G_{\upharpoonright v_0}$ be an initialized zero-sum Borel game, with $\Pi = \{1, 2\}$. Then, we have the following equality:*

$$\sup_{\sigma_1} \inf_{\sigma_2} \mu_1(\langle \bar{\sigma} \rangle_{v_0}) = \inf_{\sigma_2} \sup_{\sigma_1} \mu_1(\langle \bar{\sigma} \rangle_{v_0}).$$

That quantity is called value *of $G_{\upharpoonright v_0}$, denoted by $\mathrm{val}_1(G_{\upharpoonright v_0})$;* solving *the game G means computing its value.*

The following examples illustrate the SPE existence problem in mean-payoff games.

▶ Example 15. Let G be the mean-payoff game of Figure 1b, where each edge is labelled by its weights $\pi_{\bigcirc}$ and $\pi_{\Box}$. No weight is given for the edges ac and bd since they can be used only once, and therefore do not influence the final payoff. As shown in [9], this game does not have any SPE, neither from the state a nor from the state b.

Indeed, the only NE plays from the state b are the plays where player □ eventually leaves the cycle ab and goes to d: if he stays in the cycle ab, then player ○ would be better off leaving it, and if she does, player □ would be better off leaving it before. From the state a, if player ○ knows that player □ will leave, she has no incentive to do it before: there is no NE where ○ leaves the cycle and □ plans to do it if ever she does not. Therefore, there is no SPE where ○ leaves the cycle. But then, after a history that terminates in b, player □ has actually no incentive to leave if player ○ never plans to do it afterwards: contradiction.

▶ Example 16. Let us now study the game of Figure 2a. Using techniques from [11], we can represent the outcomes of possible plays in that game as in Figure 2b (gray and blue areas).

Following exclusively one of the three simple cycles a, ab and b of the game graph during a play yields the outcomes $01, 10$ and 22, respectively. By combining those cycles with well chosen frequencies, one can obtain any outcome in the convex hull of those three points. Now, it is also possible to obtain the point 00 by using the properties of the limit inferior: it is for instance the outcome of the play $a^2b^4a^{16}b^{256}\dots a^{2^{2^n}}b^{2^{2^{n+1}}}\dots$. In fact, one can construct a play that yields any outcome in the convex hull of the four points $00, 10, 01$, and 22.

We claim that the outcomes of SPEs plays correspond to the entire blue area in Figure 2b: there exists an SPE $\bar{\sigma}$ in $G_{\upharpoonright a}$ with $\langle \bar{\sigma} \rangle_a = \rho$ if and only if $\mu_{\Box}(\rho), \mu_{\bigcirc}(\rho) \geq 1$. That statement will be a direct consequence of the results we show in the remaining sections, but let us give a first intuition: a play with such an outcome necessarily uses infinitely often both states. It is an NE play because none of the players can get a better payoff by looping forever on their state, and they can both force each other to follow that play, by threatening them to loop for ever on their state whenever they can. But such a strategy profile is clearly not an SPE.

It can be transformed into an SPE as follows: when a player deviates, say player $\Box$, then player $\bigcirc$ can punish him by looping on a, not forever, but a great number of times, until player $\Box$'s mean-payoff gets very close to 1. Afterwards, both players follow again the play that was initially planned. Since that threat is temporary, it does not affect player $\bigcirc$'s payoff on the long term, but it really punishes player $\Box$ if that one tries to deviate infinitely often.

3 Requirements and negotiation

We will now see that SPEs are strategy profiles that respect some *requirements* about the payoffs, depending on the states it traverses. In this part, we develop the notions of *requirement* and *negotiation*.

3.1 Requirement

In the method we will develop further, we will need to analyze the players' behaviour when they have some *requirement* to satisfy. Intuitively, one can see requirements as *rationality constraints* for the players, that is, a threshold payoff value under which a player will not accept to follow a play. In all what follows, $\overline{\mathbb{R}}$ denotes the set $\mathbb{R} \cup \{\pm\infty\}$.

▶ **Definition 17** (Requirement). A *requirement* on the game G is a function $\lambda : V \to \overline{\mathbb{R}}$.

For a given state v, the quantity $\lambda(v)$ represents the minimal payoff that the player controlling v will require in a play beginning in v.

▶ **Definition 18** (λ-consistency). Let λ be a requirement on a game G. A play ρ in G is *λ-consistent* if and only if, for all $i \in \Pi$ and $n \in \mathbb{N}$ with $\rho_n \in V_i$, we have $\mu_i(\rho_n\rho_{n+1}\dots) \geq \lambda(\rho_n)$. The set of the λ-consistent plays from a state v is denoted by $\lambda\mathrm{Cons}(v)$.

▶ **Definition 19** (λ-rationality). Let λ be a requirement on a mean-payoff game G. Let $i \in \Pi$. A strategy profile $\bar{\sigma}_{-i}$ is *λ-rational* if and only if there exists a strategy σ_i such that, for every history hv compatible with $\bar{\sigma}_{-i}$, the play $\langle \bar{\sigma}_{\upharpoonright hv} \rangle_v$ is λ-consistent. We then say that the strategy profile $\bar{\sigma}_{-i}$ is λ-rational *assuming* σ_i. The set of λ-rational strategy profiles in $G_{\upharpoonright v}$ is denoted by $\lambda\mathrm{Rat}(v)$.

Note that λ-rationality is a property of a strategy profile for all the players but one, player i. Intuitively, their rationality is justified by the fact that they collectively assume that player i will, eventually, play according to the strategy σ_i: if player i does so, then everyone gets their payoff satisfied. Finally, let us define a particular requirement: the *vacuous requirement*, that requires nothing, and with which every play is consistent.

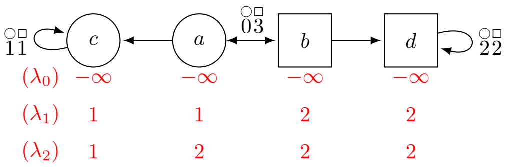


Figure 3 A game without SPE.

▶ **Definition 20** (Vacuous requirement)**.** In any game, the *vacuous requirement*, denoted by λ_0, is the requirement constantly equal to $-\infty$.

3.2 Negotiation

We will show that SPEs in prefix-independent games are characterized by the fixed points of a function on requirements. That function can be seen as a *negotiation*: when a player has a requirement to satisfy, another player can hope a better payoff than what they can secure in general, and therefore update their own requirement.

▶ **Definition 21** (Negotiation function)**.** Let G be a game. The *negotiation function* is the function that transforms any requirement λ on G into a requirement $\mathrm{nego}(\lambda)$ on G, such that for each $i \in \Pi$ and $v \in V_i$, with the convention $\inf \emptyset = +\infty$, we have:

$$\mathrm{nego}(\lambda)(v) = \inf_{\bar{\sigma}_{-i} \in \lambda\mathrm{Rat}(v)} \sup_{\sigma_i} \mu_i(\langle \bar{\sigma} \rangle_v).$$

▶ Remarks. There exists a λ-rational strategy profile from v against the player controlling v if and only if $\mathrm{nego}(\lambda)(v) \neq +\infty$. The negotiation function is monotone: if $\lambda \leq \lambda'$ (for the pointwise order, i.e. if for each v, $\lambda(v) \leq \lambda'(v)$), then $\mathrm{nego}(\lambda) \leq \mathrm{nego}(\lambda')$. The negotiation function is also non-decreasing: for every λ, we have $\lambda \leq \mathrm{nego}(\lambda)$.

In the general case, the quantity $\mathrm{nego}(\lambda)(v)$ represents the worst case value that the player controlling v can ensure, assuming that the other players play λ-rationally.

▶ Example 22. Let us consider the game of Example 15: in Figure 3, on the two first lines below the states, we present the requirements λ_0 and $\lambda_1 = \mathrm{nego}(\lambda_0)$, which is easy to compute since any strategy profile is λ_0-rational: for each v, $\lambda_1(v)$ is the classical *worst-case value* or *antagonistic value* of v, i.e. the best value the player controlling v can enforce against a fully hostile environment. Let us now compute the requirement $\lambda_2 = \mathrm{nego}(\lambda_1)$.

From c, there exists exactly one λ_1-rational strategy profile $\bar{\sigma}_{-\bigcirc} = \sigma_\Box$, which is the empty strategy since player □ has never to choose anything. Against that strategy, the best and the only payoff player ○ can get is 1, hence $\lambda_2(c) = 1$. For the same reasons, $\lambda_2(d) = 2$.

From b, player ○ can force □ to get the payoff 2 or less, with the strategy profile $\sigma_\bigcirc : h \mapsto c$. Such a strategy is λ_1-rational, assuming the strategy $\sigma_\Box : h \mapsto d$. Therefore, $\lambda_2(b) = 2$.

Finally, from a, player □ can force ○ to get the payoff 2 or less, with the strategy profile $\sigma_\Box : h \mapsto d$. Such a strategy is λ_1-rational, assuming the strategy $\sigma_\bigcirc : h \mapsto c$. But, he cannot force her to get less than the payoff 2, because she can force the access to the state b, and the only λ_1-consistent plays from b are the plays with the form $(ba)^k b d^\omega$. Therefore, $\lambda_2(a) = 2$.

3.3 Steady negotiation

In what follows, we will often need a game to be *with steady negotiation*, i.e. such that there always exists a worst λ-rational behaviour for the environment against a given player.

▶ **Definition 23** (Game with steady negotiation). A game G is *with steady negotiation* if and only if for every player i, for every vertex v, and for every requirement λ, the set $\left\{ \sup_{\sigma_i} \mu_i(\langle \bar{\sigma}_{-i}, \sigma_i \rangle_v) \mid \bar{\sigma}_{-i} \in \lambda\mathrm{Rat}(v) \right\}$ is either empty, or has a minimum.

▶ Remark. In particular, when a game is with steady negotiation, the infimum in the definition of negotiation is always reached.

It will be proved in Section 5 that mean-payoff games are with steady negotiation.

3.4 Link with Nash equilibria

Requirements and the negotiation function are able to capture Nash equilibria. Indeed, if λ_0 is the vacuous requirement, then $\mathrm{nego}(\lambda_0)$ characterizes the plays that are supported by a Nash equilibrium (abbreviated by NE plays), in the following formal sense:

▶ **Theorem 24.** *Let G be a game with steady negotiation. Then, a play ρ in G is an NE play if and only if ρ is $\mathrm{nego}(\lambda_0)$-consistent.*

▶ Example 25. Let us consider again the game of Example 15, with the requirement λ_1 given in Figure 3. The only λ_1-consistent plays in this game, starting from the state a, are ac^ω, and $(ab)^k d^\omega$ with $k \geq 1$. One can check that those plays are exactly the NE plays in that game.

In the following section, we will prove that as well as $\mathrm{nego}(\lambda_0)$ characterizes the NEs, the requirement that is the least fixed point of the negotiation function characterizes the SPEs.

4 Link between negotiation and SPEs

The notion of negotiation will enable us to find the SPEs, but also more generally the ε-SPEs, in a game. For that purpose, we need the notion of ε-fixed points of a function.

▶ **Definition 26** (ε-fixed point). Let $\varepsilon \geq 0$, let D be a finite set and let $f : \overline{\mathbb{R}}^D \to \overline{\mathbb{R}}^D$ be a mapping. A tuple $\bar{x} \in \mathbb{R}^D$ is a *ε-fixed point* of f if for each $d \in D$, for $\bar{y} = f(\bar{x})$, we have $y_d \in [x_d - \varepsilon, x_d + \varepsilon]$.

▶ Remark. A 0-fixed point is a fixed point, and conversely.

The set of requirements, equipped with the componentwise order, is a complete lattice. Since the negotiation function is monotone, Tarski's fixed point theorem states that the negotiation function has a least fixed point. That result can be generalized to ε-fixed points:

▶ **Lemma 27.** *Let $\varepsilon \geq 0$. On each game, the function* nego *has a least ε-fixed point.*

Intuitively, the ε-fixed points of the negotiation function are the requirements λ such that, from every vertex v, the player i controlling v cannot enforce a payoff greater than $\lambda(v) + \varepsilon$ against a λ-rational behaviour. Therefore, the λ-consistent plays are such that if one player tries to deviate, it is possible for the other players to prevent them improving their payoff by more than ε, while still playing rationally. Formally:

▶ **Theorem 28.** *Let $G_{\restriction v_0}$ be an initialized prefix-independent game, and let $\varepsilon \geq 0$. Let λ^* be the least ε-fixed point of the negotiation function. Let ξ be a play starting in v_0. If there exists an ε-SPE $\bar{\sigma}$ such that $\langle \bar{\sigma} \rangle_{v_0} = \xi$, then ξ is λ^*-consistent. The converse is true if the game G is with steady negotiation.*

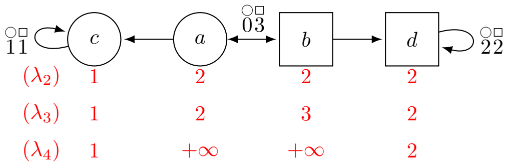


Figure 4 A game without SPE.

5 Negotiation games

We have now proved that SPEs are characterized by the requirements that are fixed points of the negotiation function; but we need to know how to compute, in practice, the quantity $\mathrm{nego}(\lambda)$ for a given requirement λ. In other words, we need a algorithm that computes, given a state v_0 controlled by a player i in the game G, and given a requirement λ, which value player i can ensure in $G_{\restriction v_0}$ if the other players play λ-rationally.

5.1 Abstract negotiation game

We first define an *abstract negotiation game*, that is conceptually simple but not directly usable for an algorithmic purpose, because it is defined on an uncoutably infinite state space.

A similar definition was given in [14], as a tool in a general method to compute SPE plays in games whose payoff functions have finite range, which is not the case of mean-payoff games. Here, linking that game with our concepts of requirements, negotiation function and steady negotiation enables us to present an effective algorithm in the case of mean-payoff games, by constructing a finite version of the abstract negotiation game, the *concrete negotiation game*, and afterwards by analyzing the negotiation function with linear algebra tools.

The abstract negotiation game from a state v_0, with regards to a player i and a requirement λ, is denoted by $\mathrm{Abs}_{\lambda i}(G)_{\restriction[v_0]}$ and opposes two players, *Prover* and *Challenger*, as follows:

- Prover proposes a λ-consistent play ρ from v_0 (or loses, if she has no play to propose).
- Then, either Challenger accepts the play and the game terminates; or, he chooses an edge $\rho_k\rho_{k+1}$, with $\rho_k \in V_i$, from which he can make player i deviate, using another edge $\rho_k v$ with $v \neq \rho_{k+1}$: then, the game starts again from v instead of v_0.
- In the resulting play (either eventually accepted by Challenger, or constructed by an infinity of deviations), Prover wants player i's payoff to be low, and Challenger wants it to be high.

That game gives us the basis of a method to compute $\mathrm{nego}(\lambda)$ from λ: the maximal outcome that Challenger – or $\mathbb{C}$ for short – can ensure in $\mathrm{Abs}_{\lambda i}(G)_{\restriction[v_0]}$, with $v_0 \in V_i$, is also the maximal payoff that player i can ensure in $G_{\restriction v_0}$, against a λ-rational environment; hence the equality $\mathrm{val}_{\mathbb{C}}\left(\mathrm{Abs}_{\lambda i}(G)_{\restriction[v_0]}\right) = \mathrm{nego}(\lambda)(v_0)$. A proof of that statement, with a complete formalization of the abstract negotiation game, is presented in [3].

▶ Example 29. Let us consider again the game of Example 15: the requirement $\lambda_2 = \mathrm{nego}(\lambda_1)$, computed in Section 3.2, is given again in Figure 4. Let us use the abstract negotiation game to compute the requirement $\lambda_3 = \mathrm{nego}(\lambda_2)$.

From a, Prover can propose the play abd^ω, and the only deviation Challenger can do is going to c; he has of course no incentive to do it. Therefore, $\lambda_3(a) = 2$. From b, whatever Prover proposes at first, Challenger can deviate and go to a. Then, from a, Prover cannot propose the play ac^ω, which is not λ_2-consistent: she has to propose a play beginning by ab, and to let Challenger deviate once more. He can then deviate infinitely often that way, and generate the play $(ba)^\omega$: therefore, $\lambda_3(b) = 3$. The other states keep the same values. Note that there exists no λ_3-consistent play from a or b, hence $\mathrm{nego}(\lambda_3)(a) = \mathrm{nego}(\lambda_3)(b) = +\infty$. This proves that there is no SPE in that game.

5.2 Concrete negotiation game

In the abstract negotiation game, Prover has to propose complete plays, on which we can make the hypothesis that they are λ-consistent. In practice, there will often be an infinity of such plays, and therefore it cannot be used directly for an algorithmic purpose. Instead, those plays can be given edge by edge, in a finite state game. Its definition is more technical, but it can be shown that it is equivalent to the abstract one. In order to make the definition as clear as possible, we give it only when the original game is a mean-payoff game. However, one could easily adapt this definition to other classes of prefix-independent games.

▶ **Definition 30** (Concrete negotiation game). Let $G_{\restriction v_0}$ be an initialized mean-payoff game, and let λ be a requirement on G, with either $\lambda(V) \subseteq \mathbb{R}$, or $\lambda = \lambda_0$.

The *concrete negotiation game* of $G_{\restriction v_0}$ for player i is the two-player zero-sum game $\mathrm{Conc}_{\lambda i}(G)_{\restriction s_0} = (\{\mathbb{P}, \mathbb{C}\}, S, (S_\mathbb{P}, S_\mathbb{C}), \Delta, \nu)_{\restriction s_0}$, defined as follows:

- The set of states controlled by Prover is $S_\mathbb{P} = V \times 2^V$, where the state $s = (v, M)$ contains the information of the current state v on which Prover has to define the strategy profile, and the *memory* M of the states that have been traversed so far since the last deviation, and that define the requirements Prover has to satisfy. The initial state is $s_0 = (v_0, \{v_0\})$.
- The set of states controlled by Challenger is $S_\mathbb{C} = E \times 2^V$, where in the state $s = (uv, M)$, the edge uv is the edge proposed by Prover.
- The set Δ contains three types of transitions: *proposals*, *acceptations* and *deviations*.
  - The proposals are transitions in which Prover proposes an edge of the game G:
$$\mathrm{Prop} = \left\{(v, M)(vw, M) \;\middle|\; vw \in E, M \in 2^V \right\};$$
  - the acceptations are transitions in which Challenger accepts to follow the edge proposed by Prover (it is in particular his only possibility when that edge begins on a state that is not controlled by player i) – note that the memory is updated:
$$\mathrm{Acc} = \{(vw, M)\,(w, M \cup \{w\}) \mid j \in \Pi, w \in V_j\};$$
  - the deviations are transitions in which Challenger refuses to follow the edge proposed by Prover, as he can if that edge begins in a state controlled by player i – the memory is erased, and only the new state the deviating edge leads to is memorized:
$$\mathrm{Dev} = \{(uv, M)(w, \{w\}) \mid u \in V_i, w \neq v, uw \in E\}.$$
- On those transitions, we define a multidimensional weight function $\hat{\pi} : \Delta \to \mathbb{R}^{\Pi \cup \{\star\}}$, with one dimension per player (*non-main* dimensions) plus one special dimension (*main* dimension) denoted by the symbol $\star$. For each non-main dimension $j \in \Pi$, we define:
  - on proposals: $\hat{\pi}_j\left((v, M)(vw, M)\right) = 0$;
  - on acceptations and deviations: $\hat{\pi}_j\left((uv, M)(w, N)\right) = 2\left(\pi_j(uw) - \max\limits_{v_j \in M \cap V_j} \lambda(v_j)\right)$;

  and on the main dimension:
  - on proposals: $\hat{\pi}_\star\left((v, M), (vw, M)\right) = 0$;
  - on acceptations and deviations: $\hat{\pi}_\star\left((uv, M), (w, N)\right) = 2\pi_i(uw)$.

For each dimension d, we write $\hat{\mu}_d$ the corresponding mean-payoff function:

$$\hat{\mu}_d(\rho) = \liminf_{n \in \mathbb{N}} \frac{1}{n} \sum_{k=0}^{n-1} \hat{\pi}_d(\rho_k \rho_{k+1}).$$

Thus, the mean-payoff along the main dimension corresponds to player i's payoff, while the mean-payoff along a non-main dimension j corresponds to player j's payoff... minus the maximal requirement player j has to satisfy.

- Then, the outcome function $\nu_{\mathbb{C}} = -\nu_{\mathbb{P}}$ measures player i's payoff, with a winning condition if the constructed strategy profile is not λ-rational, that is to say if after finitely many player i's deviations, it can generate a play which is not λ-consistent:
  - $\nu_{\mathbb{C}}(\eta) = +\infty$ if after some index $n \in \mathbb{N}$, the play $\eta_n \eta_{n+1} \ldots$ contains no deviation, and if $\hat{\mu}_j(\eta) < 0$ for some $j \in \Pi$;
  - $\nu_{\mathbb{C}}(\eta) = \hat{\mu}_\star(\eta)$ otherwise.

Like in the abstract negotiation game, the goal of Challenger is to find a λ-rational strategy profile that forces the worst possible payoff for player i, and the goal of Prover is to find a possibly deviating strategy for player i that gives them the highest possible payoff.

A play or a history in the concrete negotiation game has a projection in the game on which that negotiation game has been constructed, defined as follows:

▶ **Definition 31** (Projection of a history, of a play)**.** Let G be a prefix-independent game. Let λ be a requirement and i a player, and let $\mathrm{Conc}_{\lambda i}(G)$ be the corresponding concrete negotiation game. Let $H = (h_0, M_0)(h_0 h'_0, M_0) \ldots (h_n h'_n, M_n)$ be a history in $\mathrm{Conc}_{\lambda i}(G)$: the *projection* of the history H is the history $\dot{H} = h_0 \ldots h_n$ in the game G. That definition is naturally extended to plays.

▶ Remark. For a play η without deviations, we have $\hat{\mu}_j(\eta) \geq 0$ for each $j \in \Pi$ if and only if $\dot{\eta}$ is λ-consistent.

The concrete negotiation game is equivalent to the abstract one: the only differences are that the plays proposed by Prover are proposed edge by edge, and that their λ-consistency is not written in the rules of the game but in its outcome function.

▶ **Theorem 32.** *Let $G_{\restriction v_0}$ be an initialized mean-payoff game. Let λ be a requirement and i a player. Then, we have:*

$$\mathrm{val}_{\mathbb{C}} \left(\mathrm{Conc}_{\lambda i}(G)_{\restriction s_0}\right) = \inf_{\bar{\sigma}_{-i} \in \lambda\mathrm{Rat}(v_0)} \sup_{\sigma_i} \mu_i(\langle \bar{\sigma} \rangle_{v_0}).$$

▶ Example 33. Let us consider again the game from Example 15. Figure 5 represents the game $\mathrm{Conc}_{\lambda_1 \bigcirc}(G)$ (with $\lambda_1(a) = 1$ and $\lambda_1(b) = 2$), where the dashed states are controlled by Challenger, and the other ones by Prover. The dotted arrows indicate the deviations, and the transitions that are not labelled have either the weight 0 on the three dimensions, or meaningless weights since they cannot be used more than once. The red arrows indicate a (memoryless) optimal strategy for Challenger. Against that strategy, the lowest outcome Prover can ensure is 2. Therefore, $\mathrm{nego}(\lambda_1)(v_0) = 2$, in line with the abstract game in Example 29.

5.3 Solving the concrete negotiation game

We now know that $\mathrm{nego}(\lambda)(v)$, for a given requirement λ, a given player i and a given state $v \in V_i$, is the value of the concrete negotiation game $\mathrm{Conc}_{\lambda i}(G)_{\restriction (v, \{v\})}$. Let us now show how, in the mean-payoff case, that value can be computed.

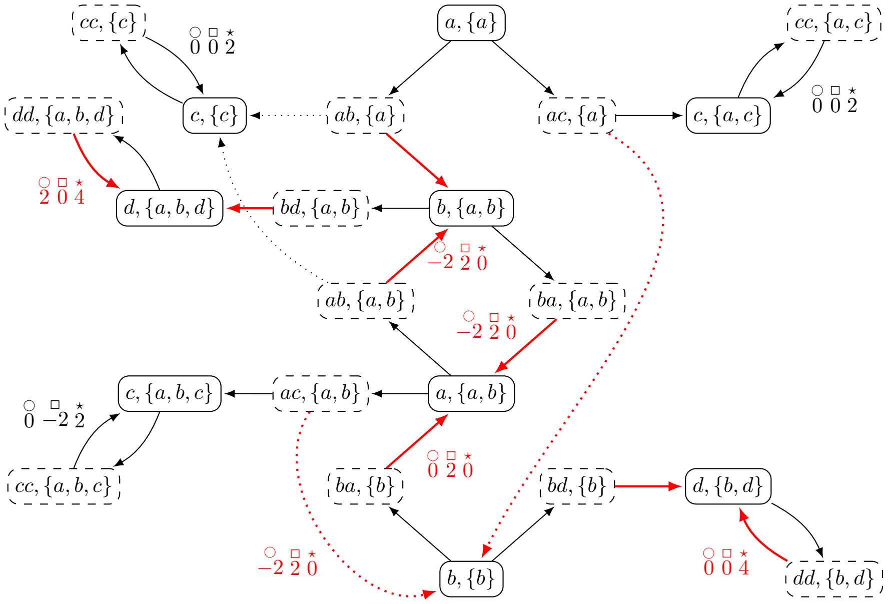


Figure 5 A concrete negotiation game.

▶ **Definition 34** (Memoryless strategy). A strategy σ_i in a game G is *memoryless* if for all vertices $v \in V_i$ and for all histories h and h', we have $\sigma_i(hv) = \sigma_i(h'v)$.

For any game G and any memoryless strategy σ_i, $G[\sigma_i]$ denotes the graph *induced* by σ_i, that is the graph (V, E'), with $E' = \{vw \in E \mid v \notin V_i \text{ or } w = \sigma_i(v)\}$. For any finite set D and any set $X \subseteq \mathbb{R}^D$, $\mathrm{Conv}X$ denotes the convex hull of X.

We can now prove that in the concrete negotiation game constructed from a mean-payoff game, Challenger has an optimal strategy that is memoryless.

▶ **Lemma 35.** *Let $G_{\restriction v_0}$ be an initialized mean-payoff game, let i be a player, let λ be a requirement and let $\mathrm{Conc}_{\lambda i}(G)_{\restriction s_0}$ be the corresponding concrete negotiation game. There exists a memoryless strategy $\tau_{\mathbb{C}}$ that is optimal for Challenger, i.e. such that:*

$$\inf_{\tau_{\mathbb{P}}} \nu_{\mathbb{C}}(\langle \bar{\tau} \rangle_{s_0}) = \mathrm{val}_{\mathbb{C}}\left(\mathrm{Conc}_{\lambda i}(G)_{\restriction s_0}\right).$$

For every game $G_{\restriction v_0}$ and each player i, $\mathrm{ML}_i\left(G_{\restriction v_0}\right)$, or $\mathrm{ML}\left(G_{\restriction v_0}\right)$ when the context is clear, denotes the set of memoryless strategies for player i in $G_{\restriction v_0}$. When (V, E) is a graph, $\mathrm{SC}(V, E)$ denotes the set of its simple cycles, and $\mathrm{SConn}(V, E)$ the set of its strongly connected components. For any closed set $C \subseteq \mathbb{R}^{\Pi \cup \{\star\}}$, the quantity $\min^\star C = \min\{x_\star \mid \bar{x} \in C, \forall j \in \Pi, x_j \geq 0\}$ is the $\star$*-minimum* of C: it will capture, in the concrete negotiation game, the least payoff that can be imposed on player i while keeping every player's payoff above their requirements, among a set of possible outcomes.

With Lemma 35, we can now solve the concrete negotiation game.

▶ **Lemma 36.** *Let $G_{\restriction v_0}$ be an initialized mean-payoff game, and let $\mathrm{Conc}_{\lambda i}(G)_{\restriction s_0}$ be its concrete negotiation game for some λ and some i. Then, the value of the game $\mathrm{Conc}_{\lambda i}(G)_{\restriction s_0}$ is given by the formula:*

$$\max_{\tau_{\mathbb{C}} \in \mathrm{ML}_{\mathbb{C}}(\mathrm{Conc}_{\lambda i}(G))} \ \min_{\substack{K \,\in\, \mathrm{SConn}\,(\mathrm{Conc}_{\lambda i}(G)[\tau_{\mathbb{C}}]) \\ \text{accessible from } s_0}} \mathrm{opt}(K),$$

where $\mathrm{opt}(K)$ is the minimal value $\nu_{\mathbb{C}}(\rho)$ for ρ among the infinite paths in K.

If K contains a deviation, then Prover can choose among its simple cycles the one that minimizes player i's payoff:

$$\mathrm{opt}(K) = \min_{c \in \mathrm{SC}(K)} \hat{\mu}_\star(c^\omega).$$

If K does not contain a deviation, then Prover must choose a combination of its simple cycles that minimizes the main dimension while keeping the other dimensions above 0*:*

$$\mathrm{opt}(K) = \min{}^\star \operatorname*{Conv}_{c \in \mathrm{SC}(K)} \hat{\mu}(c^\omega).$$

▶ **Corollary 37.** *For each player i and every state $v \in V_i$, the value* $\mathrm{nego}(\lambda)(v)$ *can be computed with the formula given in Lemma 36 applied to the game* $\mathrm{Conc}_{\lambda i}(G)_{\restriction(v,\{v\})}$

Another corollary of that result is that there always exists a best play that Prover can choose, i.e. Prover has an optimal strategy; by Theorem 32, this is equivalent to saying that:

▶ **Corollary 38.** *Mean-payoff games are games with steady negotiation.*

6 Analysis of the negotiation function in mean-payoff games

When one wants to compute the least fixed point of a function, the usual method is to iterate it on the minimal element of the considered set, to go until that fixed point. That approach is valid if the negotiation function is *Scott-continuous*, i.e. such that for every non-decreasing sequence $(\lambda_n)_n$ of requirements on G, we have $\mathrm{nego}\left(\sup_n \lambda_n\right) = \sup_n \mathrm{nego}(\lambda_n)$.

▶ **Proposition 39.** *In mean-payoff games, the negotiation function is Scott-continuous.*

By Kleene-Tarski fixed-point theorem, the least fixed point of the negotiation function is, then, the limit of the *negotiation sequence*, defined as the sequence $(\lambda_n)_{n\in\mathbb{N}} = (\mathrm{nego}^n(\lambda_0))_n$.

In many cases, the negotiation sequence is stationary, and in such a case, it is possible to compute its limit: whenever a term is equal to the previous one, we know that we reached it. But actually, the negotiation sequence is not always stationary.

▶ Example 40. Let us consider the game of Figure 6. Since all player ◇'s weights are equal to 0, for all $n > 0$, we have $\lambda_n(d) = \lambda_n(f) = 0$. It comes that for all $n > 0$, we also have $\lambda_n(c) = \lambda_n(e) = 0$. Moreover, by symmetry of the game, we always have $\lambda_n(a) = \lambda_n(b)$. Therefore, to compute the negotiation sequence, it suffices to compute $\lambda_{n+1}(a)$ as a function of $\lambda_n(b)$, knowing that $\lambda_1(a) = \lambda_1(b) = 1$, and therefore that for all $n > 0$, $\lambda_n(a) = \lambda_n(b) \geq 1$.

From a, the worst play that player □ could propose to player ○ would be a combination of the cycles cd and d giving her exactly 1. But then, player ○ will deviate to go to b, from which if player □ proposes plays in the strongly connected component containing c and d, then player ○ will always deviate and generate the play $(ab)^\omega$, and then get the payoff 2.

Then, in order to give her a payoff lower than 2, player □ has to go to the state e. Since player ○ does not control any state in that strongly connected component, the play he will propose will be accepted: he will, then, propose the worst possible combination of the cycles ef and f for player ○, such that he gets at least his requirement $\lambda_n(b)$. The payoff $\lambda_{n+1}(a)$ is then the minimal solution of the system:

$$\begin{cases} \lambda_{n+1}(a) = x + 2(1-x) \\ 2(1-x) \geq \lambda_n(b) \\ 0 \leq x \leq 1 \end{cases}$$

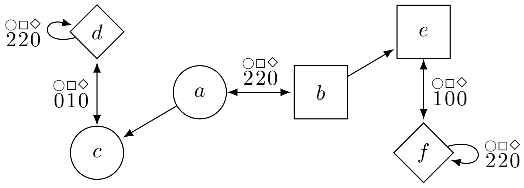
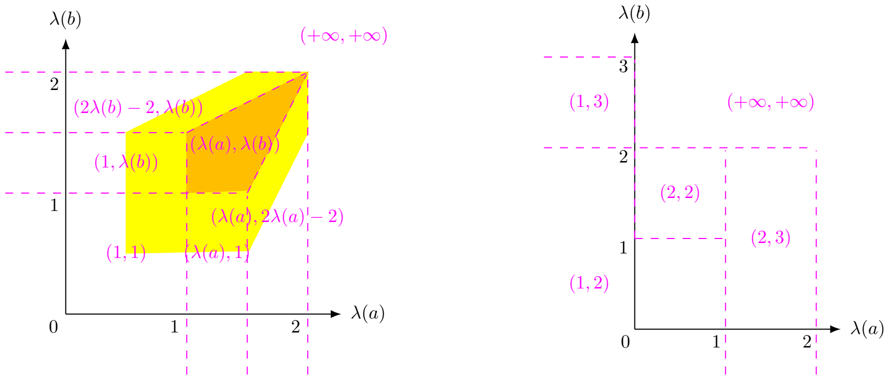


Figure 6 A game where the negotiation sequence is not stationary.



(a) Example 16.

(b) Example 15.

Figure 7 The negotiation function on the games of Examples 16 and 15.

that is to say $\lambda_{n+1}(a) = 1 + \frac{\lambda_n(b)}{2} = 1 + \frac{\lambda_n(a)}{2}$, and by induction, for all $n > 0$:

$$\lambda_n(a) = \lambda_n(b) = 2 - \frac{1}{2^{n-1}},$$

which converges to 2 but never reaches it.

Therefore, we need a different approach to compute that least fixed point. We will now show that, in the case of mean-payoff games, the negotiation function is a piecewise linear function from the vector space of requirements into itself, which can therefore be computed and analyzed using classical linear algebra techniques. Then, it becomes possible to search for the fixed points or the ε-fixed points of such a function, and to decide the existence or not of SPEs or ε-SPEs in the game studied.

▶ **Theorem 41.** *Let G be a mean-payoff game. Let us assimilate any requirement λ on G with finite values to the tuple $\bar{\lambda} = (\lambda(v))_{v \in V}$, element of the vector space $\mathbb{R}^V$. Then, for each player i and every vertex $v_0 \in V_i$, the quantity $\mathrm{nego}(\lambda)(v_0)$ is a piecewise linear function of $\bar{\lambda}$, and an effective expression of that function can be computed in 2-ExpTime.*

▶ Example 42. Let us consider the game of Example 16. If a requirement λ is represented by the tuple $(\lambda(a), \lambda(b))$, the function $\mathrm{nego} : \mathbb{R}^2 \to \mathbb{R}^2$ can be represented by Figure 7a, where in any one of the regions delimited by the dashed lines, we wrote a formula for the couple $(\mathrm{nego}(\lambda)(a), \mathrm{nego}(\lambda)(b))$. The orange area indicates the fixed points of the function, and the yellow area the other $\frac{1}{2}$-fixed points.

▶ Example 43. Now, let us consider the game of Example 15. If we fix $\lambda(c) = 1$ and $\lambda(d) = 2$, and represent the requirements λ by the tuples $(\lambda(a), \lambda(b))$, as in the previous example. Then, the negotiation function can be represented as in Figure 7b. One can check that there is no fixed point here, and even no $\frac{1}{2}$-fixed point – except $(+\infty, +\infty)$.

7 Conclusion: algorithm and complexity

Thanks to all the previous results, we are now able to compute the least fixed point, or the least ε-fixed point, of the negotiation function, on every mean-payoff game, and to use it as a characterization of all the SPEs or all the ε-SPEs. A direct application is an algorithm that solves the *ε-SPE constrained existence problem*, i.e. that decides, given an initialized mean-payoff game $G_{\upharpoonright v_0}$, two thresholds $\bar{x}, \bar{y} \in \mathbb{Q}^\Pi$, and a rational number $\varepsilon \geq 0$, whether there exists an SPE $\bar{\sigma}$ such that $\bar{x} \leq \mu(\langle \bar{\sigma} \rangle_{v_0}) \leq \bar{y}$.

We leave for future work the optimal complexity of that problem. However, we can easily prove that it cannot be solved in polynomial time, unless $\mathbf{P} = \mathbf{NP}$.

▶ **Theorem 44.** *The ε-SPE constrained existence problem is NP-hard.*

Given $G_{\upharpoonright v_0}$, by Theorem 41, computing a general expression of the negotiation function as a piecewise linear function can be done in time double exponential in the size of G. Then, for each linear piece of nego, computing its set of ε-fixed points is a polynomial problem. Since the number of pieces is at most double exponential in the size of G, computing its entire set of fixed points, and thus its least ε-fixed point λ, can be done in double exponential time.

Then, from the requirement λ and the thresholds $\bar{x}$ and $\bar{y}$, we can construct a multi-mean-payoff automaton $\mathcal{A}_\lambda$ of exponential size, that accepts an infinite word $\rho \in V^\omega$, if and only if ρ is a λ-consistent play of $G_{\upharpoonright v_0}$, and $\bar{x} \leq \mu(\rho) \leq \bar{y}$ – see [3] for the construction of $\mathcal{A}_\lambda$.

Finally, by Theorem 28, there exists an SPE $\bar{\sigma}$ in $G_{\upharpoonright v_0}$ with $\bar{x} \leq \mu(\langle \bar{\sigma} \rangle_{v_0}) \leq \bar{y}$ if and only if the language of the automaton $\mathcal{A}_\lambda$ is nonempty, which can be known in a time polynomial in the size of $\mathcal{A}_\lambda$ (see for example [1]), i.e. in a time exponential in the size of G. We can therefore conclude on the following result:

▶ **Theorem 45.** *The ε-SPE constrained existence problem is decidable and 2-ExpTime-easy.*

References

1 Rajeev Alur, Aldric Degorre, Oded Maler, and Gera Weiss. On omega-languages defined by mean-payoff conditions. In Luca de Alfaro, editor, *Foundations of Software Science and Computational Structures, 12th International Conference, FOSSACS 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, volume 5504 of *Lecture Notes in Computer Science*, pages 333–347. Springer, 2009. doi:10.1007/978-3-642-00596-1_24.

2 Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A. Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas. Non-zero sum games for reactive synthesis. In *Language and Automata Theory and Applications - 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings*, volume 9618 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2016.

3 Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. Subgame-perfect equilibria in mean-payoff games. *CoRR*, abs/2101.10685, 2021. arXiv:2101.10685.

4 Thomas Brihaye, Véronique Bruyère, Aline Goeminne, Jean-François Raskin, and Marie van den Bogaard. The complexity of subgame perfect equilibria in quantitative reachability games. In *CONCUR*, volume 140 of *LIPIcs*, pages 13:1–13:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

5 Thomas Brihaye, Véronique Bruyère, Noémie Meunier, and Jean-François Raskin. Weak subgame perfect equilibria and their application to quantitative reachability. In *24th EACSL Annual Conference on Computer Science Logic, CSL 2015, September 7-10, 2015, Berlin, Germany*, volume 41 of *LIPIcs*, pages 504–518. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

6 Thomas Brihaye, Julie De Pril, and Sven Schewe. Multiplayer cost games with simple nash equilibria. In *Logical Foundations of Computer Science, International Symposium, LFCS 2013, San Diego, CA, USA, January 6-8, 2013. Proceedings*, volume 7734 of *Lecture Notes in Computer Science*, pages 59–73. Springer, 2013.

7 Véronique Bruyère. Computer aided synthesis: A game-theoretic approach. In *Developments in Language Theory - 21st International Conference, DLT 2017, Liège, Belgium, August 7-11, 2017, Proceedings*, volume 10396 of *Lecture Notes in Computer Science*, pages 3–35. Springer, 2017.

8 Véronique Bruyère, Noémie Meunier, and Jean-François Raskin. Secure equilibria in weighted games. In *CSL-LICS*, pages 26:1–26:26. ACM, 2014.

9 Véronique Bruyère, Stéphane Le Roux, Arno Pauly, and Jean-François Raskin. On the existence of weak subgame perfect equilibria. *CoRR*, abs/1612.01402, 2016.

10 Véronique Bruyère, Stéphane Le Roux, Arno Pauly, and Jean-François Raskin. On the existence of weak subgame perfect equilibria. In *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 145–161, 2017.

11 Krishnendu Chatterjee, Laurent Doyen, Herbert Edelsbrunner, Thomas A. Henzinger, and Philippe Rannou. Mean-payoff automaton expressions. In Paul Gastin and François Laroussinie, editors, *CONCUR 2010 - Concurrency Theory, 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*, volume 6269 of *Lecture Notes in Computer Science*, pages 269–283. Springer, 2010. doi:10.1007/978-3-642-15375-4_19.

12 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. *Inf. Comput.*, 208(6):677–693, 2010. doi:10.1016/j.ic.2009.07.004.

13 János Flesch, Jeroen Kuipers, Ayala Mashiah-Yaakovi, Gijs Schoenmakers, Eilon Solan, and Koos Vrieze. Perfect-information games with lower-semicontinuous payoffs. *Math. Oper. Res.*, 35(4):742–755, 2010.

14 János Flesch and Arkadi Predtetchinski. A characterization of subgame-perfect equilibrium plays in borel games of perfect information. *Math. Oper. Res.*, 42(4):1162–1179, 2017. doi:10.1287/moor.2016.0843.

15 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016. doi:10.1007/s10472-016-9508-8.

16 Donald A. Martin. Borel determinacy. *Annals of Mathematics*, pages 363–371, 1975.

17 Noémie Meunier. *Multi-Player Quantitative Games: Equilibria and Algorithms.* PhD thesis, Université de Mons, 2016.

18 Martin J. Osborne. *An introduction to game theory.* Oxford Univ. Press, 2004.

19 Eilon Solan and Nicolas Vieille. Deterministic multi-player dynkin games. *Journal of Mathematical Economics*, 39(8):911–929, 2003.

20 Michael Ummels. Rational behaviour and strategy construction in infinite multiplayer games. In *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 212–223. Springer, 2006.

21 Nicolas Vieille and Eilon Solan. Deterministic multi-player Dynkin games. *Journal of Mathematical Economics*, Vol.39,num. 8:pp.911–929, 2003. doi:10.1016/S0304-4068(03)00021-1.

Fragility and Robustness in Mean-Payoff Adversarial Stackelberg Games

Mrudula Balachander
Université libre de Bruxelles, Brussels, Belgium

Shibashis Guha
Tata Institute of Fundamental Research, Mumbai, India

Jean-François Raskin
Université libre de Bruxelles, Brussels, Belgium

Abstract

Two-player mean-payoff Stackelberg games are nonzero-sum infinite duration games played on a bi-weighted graph by Leader (Player 0) and Follower (Player 1). Such games are played sequentially: first, Leader announces her strategy, second, Follower chooses his best-response. If we cannot impose which best-response is chosen by Follower, we say that Follower, though strategic, is adversarial towards Leader. The maximal value that Leader can get in this nonzero-sum game is called the adversarial Stackelberg value (ASV) of the game.

We study the robustness of strategies for Leader in these games against two types of deviations: (i) Modeling imprecision - the weights on the edges of the game arena may not be exactly correct, they may be delta-away from the right one. (ii) Sub-optimal response - Follower may play epsilon-optimal best-responses instead of perfect best-responses. First, we show that if the game is zero-sum then robustness is guaranteed while in the nonzero-sum case, optimal strategies for ASV are fragile. Second, we provide a solution concept to obtain strategies for Leader that are robust to both modeling imprecision, and as well as to the epsilon-optimal responses of Follower, and study several properties and algorithmic problems related to this solution concept.

2012 ACM Subject Classification Theory of computation → Solution concepts in game theory; Theory of computation → Mathematical optimization; Theory of computation → Logic and verification

Keywords and phrases mean-payoff, Stackelberg games, synthesis

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.9

Related Version *Full Version*: https://arxiv.org/pdf/2007.07209.pdf

Funding This work is partially supported by the PDR project Subgame perfection in graph games (F.R.S.-FNRS), the ARC project Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond (Fédération Wallonie-Bruxelles), the EOS project Verifying Learning Artificial Intelligence Systems (F.R.S.-FNRS and FWO), and the COST Action 16228 GAMENET (European Cooperation in Science and Technology).

1 Introduction

Stackelberg games [16] were first introduced to model strategic interactions among rational agents in markets that consist of Leader and Follower(s). Leader in the market makes her strategy public and Follower(s) respond by playing an optimal response to this strategy. Here, we consider Stackelberg games as a framework for the synthesis of reactive programs [15, 3]. These programs maintain a continuous interaction with the environment in which they operate; they are *deterministic functions* that given a history of interactions so far choose an action. Our work is a contribution to *rational synthesis* [9, 14], a nonzero-sum game setting where both the program and the environment are considered as rational agents that have their own goals. While Boolean ω-regular payoff functions have been studied in [9, 14], here we study the quantitative long-run average (mean-payoff) function.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 9; pp. 9:1–9:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

We illustrate our setting with the example in Figure 1. The set V of vertices is partitioned into V_0 (represented by circles) and V_1 (represented by squares) that are owned by Leader (also called Player 0) and Follower (also called Player 1) respectively. In the tuple on the edges, the first element is the payoff of Leader, while the second one is the payoff of Follower (weights are omitted if they are both equal to 0). Each player's objective is to maximize the long run average of the payoffs that she receives (a.k.a. mean-payoff). In the adversarial Stackelberg setting, Player 0 (Leader) first announces how she will play then Player 1 (Follower) chooses one of his best-responses to this strategy. Here, there are two choices for Player 0: L or R. As Player 1 is assumed to be rational, Player 0 deduces that she must play L. Indeed, the best response of Player 1 is then to play LL and the reward she obtains is 10. This is better than playing R, for which the best-response of Player 1 is RL, and the reward is 8 instead of 10. Note that if there are several possible best responses for Player 1, then we consider the worst-case: Player 0 has no control on the choice of best-responses by Player 1.

Quantitative models and robustness. The study of adversarial Stackelberg games with mean-payoff objectives has been started in [8] with the concept of *adversarial Stackelberg value* (ASV for short). ASV is the best value that Leader can obtain by fixing her strategy and facing any rational response by Follower. As this setting is quantitative, it naturally triggers questions about *robustness* that were left open in this first paper.

Robustness is a *highly desirable* property of quantitative models: small changes in the quantities appearing in a model M (e.g. rewards, probabilities, etc.) should have small impacts on the predictions made from M, see e.g. [2]. Robustness is thus crucial because it accounts for modelling imprecision that are inherent in quantitative modelling and those imprecision may have important consequences. For instance, a reactive program synthesized from a model M should provide acceptable performances if it is executed in a real environment that differ slightly w.r.t. the quantities modeled in M.

Some classes of models are robust. For instance, consider two-player *zero-sum* mean-payoff games where players have fully antagonistic objectives. The *value* of a two-player zero-sum mean-payoff $\mathcal{G}$ is the maximum mean-payoff that Player 0 can ensure against all strategies of Player 1. A strategy σ_0 that enforces the optimal value c in $\mathcal{G}$ is *robust* in the following sense. Let $\mathcal{G}^{\pm\delta}$ be the set of games obtained by increasing or decreasing the weights on the edges of $\mathcal{G}$ by at most δ. Then for all $\delta > 0$, and for all $\mathcal{H} \in \mathcal{G}^{\pm\delta}$, the strategy σ_0 ensures in $\mathcal{H}$ a mean-payoff of at least $c - \delta$ for Player 0 against any strategy of Player 1 (Proposition 1). So slight changes in the quantities appearing in the model have only a small impact on the worst-case value enforced by the strategy.

The situation is more complex and less satisfactory in *nonzero-sum* games. Strategies that enforce the ASV proposed in [8] may be *fragile*: slight differences in the weights of the game, or in the optimality of the response by Player 1, may lead to large differences in the value obtained by the strategy. We illustrate these difficulties on our running example. The strategy of Player 0 that chooses L in v_0 ensures her a payoff of 10 which is the ASV. Indeed, the unique best-response of Player 1 against L is to play LL from v_1. However, if the weights in $\mathcal{G}$ are changed by up to $\pm\delta = \pm 0.6$ then there is a game $\mathcal{H} \in \mathcal{G}^{\pm\delta}$ in which the weight on the self-loop over vertex v_4 changes to e.g. 9.55, and the weight on the self-loop over v_3 changes to e.g. 9.45, and the action LR becomes better for Player 1. So the value of L in $\mathcal{H}$ against a rational adversary is now 0 instead of 10. Thus a slight change in the rewards for Player 1 (due to e.g. modelling imprecision) may have a dramatic effect on the value of the optimal strategy L computed on the model $\mathcal{G}$ when evaluated in $\mathcal{H}$.

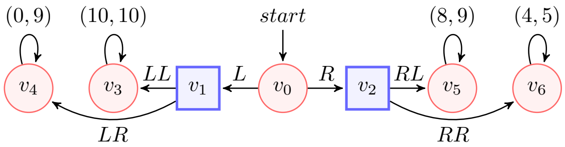


Figure 1 A game in which the strategy of Leader that maximizes the adversarial Stackelberg is fragile while the strategy of Leader that maximizes the $\epsilon = 1$-adversarial Stackelberg value is robust.

Contributions. As a remedy to this situation, we provide an alternative notion of value that is better-suited to synthesize strategies that are robust against perturbations. We consider two types of perturbations. First, the strategies computed for this value are robust against *modeling imprecision*: if a strategy has been synthesized from a weighted game graph with weights that are possibly slightly wrong, the value that this strategy delivers is guaranteed to be close to what the model predicts. Second, strategies computed for this value are robust against *sub-optimal responses*: small deviations from the best-response by the adversary have only limited effect on the value guaranteed by the strategy.

Our solution relies on relaxing the notion of best-responses of Player 1 *in the original model* $\mathcal{G}$: we define the ϵ-adversarial Stackelberg value (ASV^ϵ, for short) as the value that Leader can enforce against all ϵ-best responses of Follower. Obviously, this directly accounts for the second type of perturbations. But we show that, additionally, this accounts for the first type of perturbations: if a strategy σ_0 enforces an ASV^ϵ equal to c then for all games $\mathcal{H} \in G^{\pm\frac{\epsilon}{2}}$, we have that σ_0 enforce a value larger than $c - \epsilon$ in $\mathcal{H}$ (Theorem 5 and Theorem 6).

We illustrate this by considering again the example of Figure 1. Here, if we consider that the adversary can play $2\delta = 1.2$-best responses instead of best responses only, then the optimal strategy of Player 0 is now R and it has a ASV^ϵ equal to 8. This value is guaranteed to be robust for all games $\mathcal{H} \in \mathcal{G}^{\pm\delta}$ as R is guaranteed to enforce a payoff that is larger than $8 - \delta$ in all games in $\mathcal{G}^{\pm\delta}$. Stated otherwise, we use the notion of ASV^ϵ in the original game to find a strategy for Player 0 that she uses in the perturbed model while playing against a rational adversary. Thus we show that in the event of modelling imprecision resulting in a perturbed model, the solution concept to be used is ASV^ϵ instead of ASV since the former provides strategies that are robust to such perturbations.

Table 1 Summary of our results.

	Robustness	Threshold Problem	Computing ASV	Achievability
Adversarial best responses of Follower	**No** **[Proposition 2]**	NP [8] **Finite Memory Strategy [1]** **Memoryless Strategy [1]**	Theory of Reals [8]	No [8]
Adversarial ϵ-best responses of Follower	**Yes** **[Thm 5]**	**NP** **Finite Memory Strategy [Thm 8]** **Memoryless Strategy [Thm 10]**	**Theory of Reals [Thm 12]** **Solving LP in EXPTime [Thm 12]**	**Yes [Thm 16]** **(Requires Infinite Memory [Thm 19])**

In addition to proving the fragility of the original concept introduced in [8] (Proposition 2) and the introduction of the new notion of value ASV^ϵ that is robust against modelling imprecision (Theorem 5), we provide algorithms to handle ASV^ϵ. First, we show how to decide the threshold problem for ASV^{ϵ} in nondeterministic polynomial time and that finite memory strategies suffice (Theorem 8). Second, we provide an algorithm to compute ASV^{ϵ} when ϵ is fixed (Theorem 12). Third, we provide an algorithm that given a threshold value c, computes the largest possible ϵ such that $\mathsf{ASV}^{\epsilon} > c$ (Corollary 15). These three results form the core technical contributions of this paper and they are presented in Section 4 and Section 5. Additionally, in Section 6, we show that ASV^{ϵ} is always achievable (Theorem 16), which is in contrast to the case in [8] where Follower only plays best-responses. Finally, we provide results that concern the memory needed for players to play optimally, and complexity results for subcases (for example when Players are assumed to play memoryless). Our contributions have been summarized in Table 1, where the results obtained in this work are in bold. The new results corresponding to ASV can be found in [1].

Related Works. Stackelberg games on graphs have been first considered in [9], where the authors study rational synthesis for ω-regular objectives with co-operative Follower(s). In [8], Stackelberg mean-payoff games in adversarial setting, and Stackelberg discounted sum games in both adversarial and co-operative setting have been considered. However, as pointed out earlier, the model of [8] is not robust to perturbations. In [10], mean-payoff Stackelberg games in the co-operative setting have been studied. In [13], the authors study the effects of limited memory on both Nash and Stackelberg (or leader) strategies in multi-player discounted sum games. Incentive equilibrium over bi-matrix games and over mean-payoff games in a co-operative setting have been studied in [11] and [12] respectively. In [14], adversarial rational synthesis for ω-regular objectives have been studied. In [7], precise complexity results for various ω-regular objectives have been established for both adversarial and co-operative settings. In [6, 4], secure Nash equilibrium has been studied, where each player first maximises her own payoff, and then minimises the payoff of the other player; Player 0 and Player 1 are symmetric there unlike in Stackelberg games. For discounted sum objectives, in [8], the gap problem has been studied. Given rationals c and δ, a solution to the gap problem can decide if $\mathsf{ASV} > c + \delta$ or $\mathsf{ASV} < c - \delta$. The threshold problem was left open in [8], and is technically challenging. We leave the case of analysing robustness for discounted sum objective for future work.

A full version of this work with detailed proofs appears in [1].

2 Preliminaries

We denote by $\mathbb{N}$, $\mathbb{N}^{+}$, $\mathbb{Q}$, and $\mathbb{R}$ the set of naturals, the set of naturals excluding 0, the set of rationals, and the set of reals respectively.

Arenas. An (bi-weighted) arena $\mathcal{A} = (V, E, \langle V_0, V_1\rangle, w_0, w_1)$ consists of a finite set V of vertices, a set $E \subseteq V \times V$ of edges such that for all $v \in V$ there exists $v' \in V$ and $(v, v') \in E$, a partition $\langle V_0, V_1\rangle$ of V, where V_0 (resp. V_1) is the set of vertices for Player 0 (resp. Player 1), and two edge weight functions $w_0 : E \to \mathbb{Z}$, $w_1 : E \to \mathbb{Z}$. In the sequel, we denote the maximum absolute value of a weight in $\mathcal{A}$ by W. A strongly connected component of a directed graph is a subgraph that is strongly connected. Unless otherwise mentioned, *SCC* denotes a subgraph that is strongly connected, and which may or may not be maximal.

Plays and histories. A play in $\mathcal{A}$ is an infinite sequence of vertices $\pi = \pi_0\pi_1 \cdots \in V^{\omega}$ such that for all $k \in \mathbb{N}$, we have $(\pi_k, \pi_{k+1}) \in E$. A *history* in $\mathcal{A}$ is a (non-empty) prefix of a play in $\mathcal{A}$. Given $\pi = \pi_0\pi_1 \cdots \in \mathsf{Plays}_{\mathcal{A}}$ and $k \in \mathbb{N}$, the prefix $\pi_0\pi_1 \ldots \pi_k$ of π is denoted by $\pi_{\leqslant k}$.

We denote by $\inf(\pi)$ the set of vertices v that appear infinitely many times along π, i.e., $\inf(\pi) = \{v \in V \mid \forall i \in \mathbb{N} \cdot \exists j \in \mathbb{N}, j \geqslant i : \pi(j) = v\}$. It is easy to see that $\inf(\pi)$ forms an SCC in the underlying graph of the arena $\mathcal{A}$. We denote by $\mathsf{Plays}_{\mathcal{A}}$ and $\mathsf{Hist}_{\mathcal{A}}$ the set of plays and the set of histories in $\mathcal{A}$ respectively; the symbol $\mathcal{A}$ is omitted when clear from the context. Given $i \in \{0,1\}$, the set $\mathsf{Hist}^i_{\mathcal{A}}$ denotes the set of histories such that their last vertex belongs to V_i. We denote the first vertex and the last vertex of a history h by $\mathsf{first}(h)$ and $\mathsf{last}(h)$ respectively.

Games. A *mean-payoff game* $\mathcal{G} = (\mathcal{A}, \langle \underline{\mathsf{MP}}_0, \underline{\mathsf{MP}}_1\rangle)$ consists of a bi-weighted arena $\mathcal{A}$, payoff functions $\underline{\mathsf{MP}}_0 : \mathsf{Plays}_{\mathcal{A}} \to \mathbb{R}$ and $\underline{\mathsf{MP}}_1 : \mathsf{Plays}_{\mathcal{A}} \to \mathbb{R}$ for for Player 0 and Player 1 respectively which are defined as follows. Given a play $\pi \in \mathsf{Plays}_{\mathcal{A}}$ and $i \in \{0,1\}$, the payoff $\underline{\mathsf{MP}}_i(\pi)$ is given by $\underline{\mathsf{MP}}_i(\pi) = \liminf\limits_{k\to\infty} \frac{1}{k} w_i(\pi_{\leqslant k})$, where the weight $w_i(h)$ of a history $h \in \mathsf{Hist}$ is the sum of the weights assigned by w_i to its edges. In our definition of the mean-payoff, we have used $\liminf$ as the limit of the successive average may not exist.

Strategies and payoffs. A strategy for Player $i \in \{0,1\}$ in the game $\mathcal{G}$ is a function $\sigma : \mathsf{Hist}^i_{\mathcal{A}} \to V$ that maps histories ending in a vertex $v \in V_i$ to a successor of v. The set of all strategies of Player $i \in \{0,1\}$ in the game $\mathcal{G}$ is denoted by $\Sigma_i(\mathcal{G})$, or Σ_i when $\mathcal{G}$ is clear from the context. A strategy has memory M if it can be realized as the output of a state machine with M states. A memoryless strategy is a function that only depends on the last element of the history $h \in \mathsf{Hist}$. We denote by Σ_i^{ML} the set of memoryless strategies of Player i, and by Σ_i^{FM} her set of finite memory strategies. A *profile* is a pair of strategies $\overline{\sigma} = (\sigma_0, \sigma_1)$, where $\sigma_0 \in \Sigma_0(\mathcal{G})$ and $\sigma_1 \in \Sigma_1(\mathcal{G})$. As we consider games with perfect information and deterministic transitions, any profile $\overline{\sigma}$ yields, from any history h, a unique play or *outcome*, denoted $\mathsf{Out}_h(\mathcal{G}, \overline{\sigma})$. Formally, $\mathsf{Out}_h(\mathcal{G}, \overline{\sigma})$ is the play π such that $\pi_{\leqslant |h|-1} = h$ and $\forall k \geqslant |h| - 1$ it holds that $\pi_{k+1} = \sigma_i(\pi_{\leqslant k})$ if $\pi_k \in V_i$. We write $h \leqslant \pi$ whenever h is a prefix of π. The set of outcomes compatible with a strategy $\sigma \in \Sigma_{i\in\{0,1\}}(\mathcal{G})$ after a history h is $\mathsf{Out}_h(\mathcal{G}, \sigma) = \{\pi | \exists \sigma' \in \Sigma_{1-i}(\mathcal{G})$ such that $\pi = \mathsf{Out}_h(\mathcal{G}, (\sigma, \sigma'))\}$. Each outcome $\pi \in \mathcal{G} = (\mathcal{A}, \langle \underline{\mathsf{MP}}_0, \underline{\mathsf{MP}}_1\rangle)$ yields a payoff $\underline{\mathsf{MP}}(\pi) = (\underline{\mathsf{MP}}_0(\pi), \underline{\mathsf{MP}}_1(\pi))$.

Usually, we consider instances of games such that the players start playing at a fixed vertex v_0. Thus, we call an initialized game a pair $(\mathcal{G}, v_0)$, where $\mathcal{G}$ is a game and $v_0 \in V$ is the initial vertex. When v_0 is clear from context, we use $\mathcal{G}$, $\mathsf{Out}(\mathcal{G}, \overline{\sigma})$, $\mathsf{Out}(\mathcal{G}, \sigma)$, $\underline{\mathsf{MP}}(\overline{\sigma})$ instead of $\mathcal{G}_{v_0}$, $\mathsf{Out}_{v_0}(\mathcal{G}, \overline{\sigma})$, $\mathsf{Out}_{v_0}(\mathcal{G}, \sigma)$, $\underline{\mathsf{MP}}_{v_0}(\overline{\sigma})$. We sometimes omit $\mathcal{G}$ when it is clear from the context.

Best-responses, ϵ-best-responses. Let $\mathcal{G} = (\mathcal{A}, \langle \underline{\mathsf{MP}}_0, \underline{\mathsf{MP}}_1\rangle)$ be a two-dimensional mean-payoff game on the bi-weighted arena $\mathcal{A}$. Given a strategy σ_0 for Player 0, we define

1. Player 1's best responses to σ_0, denoted by $\mathsf{BR}_1(\sigma_0)$, as:

$$\{\sigma_1 \in \Sigma_1 \mid \forall v \in V. \forall \sigma_1' \in \Sigma_1 : \mathsf{MP}_1(\mathsf{Out}_v(\sigma_0, \sigma_1)) \geqslant \mathsf{MP}_1(\mathsf{Out}_v(\sigma_0, \sigma_1'))\}$$

2. Player 1's ϵ-best-responses to σ_0, for $\epsilon > 0$[1], denoted by $\mathsf{BR}_1^{\epsilon}(\sigma_0)$, as:

$$\{\sigma_1 \in \Sigma_1 \mid \forall v \in V \cdot \forall \sigma_1' \in \Sigma_1 : \underline{\mathsf{MP}}_1(\mathsf{Out}_v(\sigma_0, \sigma_1)) > \underline{\mathsf{MP}}_1(\mathsf{Out}_v(\sigma_0, \sigma_1')) - \epsilon\}$$

[1] Since we will use ϵ in ASV^{ϵ} to add robustness, we only consider the cases in which ϵ is strictly greater than 0.

We also introduce the following notation for zero-sum games (that are needed as intermediary steps in our algorithms). Let $\mathcal{A}$ be an arena, $v \in V$ one of its states, and $\mathcal{O} \subseteq \mathsf{Plays}_{\mathcal{A}}$ be a set of plays (called objective), then we write $\mathcal{A}, v \models\ll i \gg \mathcal{O}$, if:

$$\exists \sigma_i \in \Sigma_i \cdot \forall \sigma_{1-i} \in \Sigma_{1-i} : \mathsf{Out}_v(\mathcal{A}, (\sigma_i, \sigma_{1-i})) \in \mathcal{O}, \text{for } i \in \{0,1\}$$

All the zero-sum games we consider in this paper are *determined* meaning that for all $\mathcal{A}$, for all objectives $\mathcal{O} \subseteq \mathsf{Plays}_{\mathcal{A}}$ we have that $\mathcal{A}, v \models\ll i \gg \mathcal{O} \iff \mathcal{A}, v \not\models\ll 1-i \gg \mathsf{Plays}_{\mathcal{A}} \setminus \mathcal{O}$. We sometimes omit $\mathcal{A}$ when the arena being referenced is clear from the context.

Convex hull and $\mathsf{F}_{\mathbf{min}}$. Given a finite dimension d, a finite set $X \subset \mathbb{Q}^d$ of rational vectors, we define the convex hull $\mathsf{CH}(X) = \{v \mid v = \sum_{x \in X} \alpha_x \cdot x \wedge \forall x \in X : \alpha_x \in [0,1] \wedge \sum_{x \in X} \alpha_x = 1\}$ as the set of all their convex combinations. Let $f_{\min}(X)$ be the vector $v = (v_1, v_2, \dots, v_d)$ where $v_i = \min\{c \mid \exists x \in X : x_i = c\}$ i.e. the vector v is the pointwise minimum of the vectors in X. For $S \subseteq \mathbb{Q}^d$, we define $\mathsf{F}_{\min}(S) = \{f_{\min}(P) \mid P \text{ is a finite subset of } S\}$.

Mean-payoffs induced by simple cycles. A *cycle* c is a sequence of edges that starts and stops in a given vertex v, it is simple if it does not contain repetition of any other vertex. Given an SCC S, we write $\mathbb{C}(S)$ for the set of simple cycles inside S. Given a simple cycle c, for $i \in \{0,1\}$, let $\mathsf{MP}_i(c) = \frac{w_i(c)}{|c|}$ be the mean of the weights[2] in each dimension along the edges in the simple cycle c, and we call the pair $(\mathsf{MP}_0(c), \mathsf{MP}_1(c))$ the mean-payoff coordinate of the cycle c. We write $\mathsf{CH}(\mathbb{C}(S))$ for the convex-hull of the set of mean-payoff coordinates of simple cycles of S.

Adversarial Stackelberg Value for MP. Since the set of best-responses in mean-payoff games can be empty (See Lemma 3 of [8]), we use the notion of ϵ-best-responses for the definition of ASV which are guaranteed to always exist[3]. We define

$$\mathsf{ASV}(v) = \sup_{\sigma_0 \in \Sigma_0, \epsilon > 0} \inf_{\sigma_1 \in \mathsf{BR}_1^{\epsilon}(\sigma_0)} \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1)).^4$$

We also associate a (adversarial) value to a strategy $\sigma_0 \in \Sigma_0$ of Player 0, denoted

$$\mathsf{ASV}(\sigma_0)(v) = \sup_{\epsilon > 0} \inf_{\sigma_1 \in \mathsf{BR}_1^{\epsilon}(\sigma_0)} \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1)).$$

Clearly, we have that $\mathsf{ASV}(v) = \sup_{\sigma_0 \in \Sigma_0} \mathsf{ASV}(\sigma_0)(v)$.

We define the adversarial Stackelberg values, where strategies of Player 0 are restricted to finite memory strategies, as

$$\mathsf{ASV}_{\mathsf{FM}}(v) = \sup_{\sigma_0 \in \Sigma_0^{\mathsf{FM}}} \inf_{\sigma_1 \in \mathsf{BR}_1(\sigma_0)} \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1))$$

where Σ_0^{FM} refers to the set of all finite memory strategies of Player 0. We note that for every finite memory strategy σ_0 of Player 0, a best-response of Player 1 to σ_0 always exists as noted in [8].

We also define the adversarial Stackelberg values, where Player 0 is restricted to using memoryless strategies, as

$$\mathsf{ASV}_{\mathsf{ML}}(v) = \sup_{\sigma_0 \in \Sigma_0^{\mathsf{ML}}} \inf_{\sigma_1 \in \mathsf{BR}_1(\sigma_0)} \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1))$$

where Σ_0^{ML} is the set of all memoryless strategies of Player 0.

[2] We do not use $\underline{\mathsf{MP}}_i$ since lim inf and lim sup are the same for a finite sequence of edges.

[3] For a game $\mathcal{G}$, we also use $\mathsf{ASV}_{\mathcal{G}}$ and $\mathsf{ASV}^{\epsilon}_{\mathcal{G}}$, and drop the subscript $\mathcal{G}$ when it is clear from the context.

[4] The definition of ASV, as it appears in [8], is syntactically different but the two definitions are equivalent, and the one presented here is simpler.

In the sequel, unless otherwise mentioned, we refer to a two-dimensional nonzero-sum two-player mean-payoff game simply as a mean-payoff game.

Zero-sum case. Zero-sum games are special cases of nonzero-sum games, where for all edges $e \in E$, we have that $w_0(e) = -w_1(e)$, i.e. the gain of one player is always equal to the opposite (the loss) of the other player. For zero-sum games, the classical concept is the notion of (worst-case) value. It is defined as

$$\mathsf{Val}_{\mathcal{G}}(v) = \sup_{\sigma_0 \in \Sigma_0} \inf_{\sigma_1 \in \Sigma_1} \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1)).$$

Additionally, we define the *value* of a Player 0 strategy σ_0 from a vertex v in a zero-sum mean-payoff game $\mathcal{G}$ as $\mathsf{Val}_{\mathcal{G}}(\sigma_0)(v) = \inf_{\sigma_1 \in \Sigma_1} \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1))$.

3 Fragility and robustness in games

In this section, we study fragility and robustness properties in *zero-sum* and *nonzero-sum* games. Additionally, we provide a notion of value, for the nonzero-sum case, that is well-suited to synthesize strategies that are robust against two types of perturbations:

- Modeling imprecision: We want guarantees about the value that is obtained by a strategy in the Stackelberg game even if this strategy has been synthesized from a weighted game graph with weights that are possibly slightly wrong: small perturbations of the weight should have only limited effect on the value guaranteed by the strategy.
- Sub-optimal responses: We want guarantees about the value that is obtained by a strategy in the Stackelberg game even if the adversary responds with an ϵ-best response instead of a perfectly optimal response (for some $\epsilon > 0$): small deviations from the best-response by the adversary should have only limited effect on the value guaranteed by the strategy.

Formalizing deviations. To formalize modeling imprecision, we introduce the notion of a *perturbed game graph.* Given a game $\mathcal{G}$ with arena $\mathcal{A}_{\mathcal{G}} = (V, E, \langle V_0, V_1 \rangle, w_0, w_1)$, and a value $\delta > 0$, we write $\mathcal{G}^{\pm\delta}$ for the set $\mathcal{H}$ of games with arena $\mathcal{A}_{\mathcal{H}} = (V, E, \langle V_0, V_1 \rangle, w_0', w_1')$ where edge weight functions respect the following constraints:

$$\forall (v_1, v_2) \in E, \forall i \in \{0, 1\}, \quad w_i'(v_1, v_2) \in (w_i(v_1, v_2) + \delta, w_i(v_1, v_2) - \delta).$$

We note that as the underlying game graph (V, E) is not altered, for both players, the set of strategies in $\mathcal{G}$ is identical to the set of strategies in $\mathcal{H}$. Finally, to formalize *sub-optimal responses*, we naturally use the notion of ϵ-best response introduced in the previous section.

Robustness in zero-sum games. In zero-sum games, the worst-case value $\mathsf{Val}_{\mathcal{G}}(\sigma_0)$ is robust against both modeling imprecision and sub-optimal responses of Player 1.

▶ **Proposition 1** (Robustness in zero-sum games). *For all zero-sum mean-payoff games $\mathcal{G}$ with a set V of vertices, for all Player 0 strategies σ_0,and for all vertices $v \in V$ we have that:*

$$\forall \delta, \epsilon > 0 : \forall \mathcal{H} \in \mathcal{G}^{\pm\delta} : \inf_{\sigma_1 \in \mathsf{BR}^{\epsilon}_{1,\mathcal{H}}(\sigma_0)} \underline{\mathsf{MP}}^{\mathcal{H}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1)) > \mathsf{Val}_{\mathcal{G}}(\sigma_0)(v) - \delta.$$

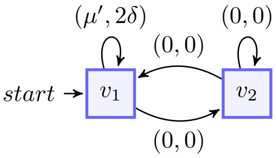


Figure 2 In this game $\mathcal{G}$, all vertices are controlled by Player 1. Here, $\mathsf{ASV}_{\mathcal{G}}(v_1) = \mu'$.

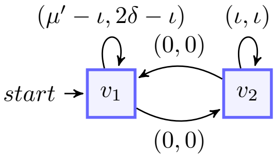


Figure 3 An δ-perturbed game $\mathcal{H}$ of $\mathcal{G}$ in Figure 2. Here, we consider $0 < \iota < \delta$. Here, $\mathsf{ASV}_{\mathcal{H}}(v_1) = \mu' - \iota$.

Fragility in non-zero sum games. On the contrary, the adversarial Stackelberg value $\mathsf{ASV}(\sigma_0)$ is fragile against both modeling imprecision and sub-optimal responses.

▶ **Proposition 2** (Fragility - modeling imprecision)**.** *For all $\mu > 0$, we can construct a nonzero-sum mean-payoff game $\mathcal{G}$ and a Player 0 strategy σ_0, such that there exist $\delta > 0$, a perturbed game $\mathcal{H} \in \mathcal{G}^{\pm\delta}$, and a vertex v in $\mathcal{G}$ with $\mathsf{ASV}_{\mathcal{H}}(\sigma_0)(v) < \mathsf{ASV}_{\mathcal{G}}(\sigma_0)(v) - \mu$.*

▶ **Proposition 3** (Fragility - sub-optimal responses)**.** *For all $\mu > 0$, we can construct a nonzero-sum mean-payoff game $\mathcal{G}$ and a Player 0 strategy σ_0, such that there exist $\epsilon > 0$ and a vertex v in $\mathcal{G}$ with $\inf_{\sigma_1 \in \mathsf{BR}_1^{\epsilon}(\sigma_0)} \underline{\mathsf{MP}}_0^{\mathcal{G}}(\mathsf{Out}_v(\sigma_0, \sigma_1)) < \mathsf{ASV}_{\mathcal{G}}(\sigma_0)(v) - \mu$.*

Note that μ can be arbitrarily large and thus the adversarial Stackelberg value in the model under deviations can be arbitrarily worse than in the original model.

Relation between the two types of deviations. In nonzero-sum mean-payoff games, robustness against modeling imprecision does not imply robustness against sub-optimal responses.

▶ **Lemma 4.** *For all $\mu, \delta, \epsilon > 0$, we can construct a nonzero-sum mean-payoff game $\mathcal{G}$ such that for all Player 0 strategies σ_0 and vertex v in $\mathcal{G}$, we have that:*

$$\forall \mathcal{H} \in \mathcal{G}^{\pm\delta} : \mathsf{ASV}_{\mathcal{H}}(\sigma_0)(v) > \inf_{\sigma_1 \in \mathsf{BR}_{1,\mathcal{G}}^{\epsilon}} \underline{\mathsf{MP}}_0^{\mathcal{G}}(\mathsf{Out}_v(\sigma_0, \sigma_1)) + \mu.$$

Proof. Consider the game $\mathcal{G}$ shown in Figure 2. Here, since all the vertices are controlled by Player 1, the strategy of Player 0 is inconsequential. For every $\delta > 0$, we claim that the best strategy for Player 1 across all perturbed games $\mathcal{H} \in \mathcal{G}^{\pm\delta}$ is to play $v_1 \to v_1$ forever. One such example of a perturbed game is shown in Figure 3. Here, for every $0 < \iota < \delta$, we have that $v_1 \to v_1$ is the only best-response for Player 1. Therefore, we have that $\inf_{\mathcal{H} \in \mathcal{G}^{\pm\delta}} \mathsf{ASV}_{\mathcal{H}}(\sigma_0)(v_1) = \mu' - \delta$, for all $\delta > 0$.

However, if we relax the assumption that Player 1 plays optimally and assume that he plays an ϵ-best response in the game $\mathcal{G}$, we note that Player 1 can play a strategy $(v_1^{k_1+1} v_2^{k_2+1})^{\omega}$, for some $k_1, k_2 \in \mathbb{N}$, such that $\frac{2\delta \cdot k_1}{k_1 + k_2 + 2} > 2\delta - \epsilon$, and Player 0 gets a payoff of $\frac{k_1 \cdot \mu'}{k_1 + k_2 + 2} > \mu'(1 - \frac{\epsilon}{2\delta})$. Thus, we have that $\inf_{\sigma_1 \in \mathsf{BR}_{1,\mathcal{G}}^{\epsilon}} \underline{\mathsf{MP}}_0^{\mathcal{G}}(\mathsf{Out}_v(\sigma_0, \sigma_1)) = \mu'(1 - \frac{\epsilon}{2\delta})$. We note that the choice of μ' is arbitrary, and we can have a μ' such that $\mu' - \delta > \mu'(1 - \frac{\epsilon}{2\delta}) + \mu$, i.e, we choose μ' to be large enough so that $\mu < \mu' \cdot \frac{\epsilon}{2\delta} - \delta$. ◀

On the contrary, robustness against sub-optimal responses implies robustness against modeling imprecision.

▶ **Theorem 5** (Robust strategy in non-zero sum games). *For all non-zero sum mean-payoff games $\mathcal{G}$ with a set V of vertices, for all $\epsilon > 0$, for all vertices $v \in V$, for all strategies σ_0 of Player 0, we have that $\forall \mathcal{H} \in \mathcal{G}^{\pm\epsilon} : \mathsf{ASV}_{\mathcal{H}}(\sigma_0)(v) > \inf_{\sigma_1 \in \mathsf{BR}^{2\epsilon}_{1,\mathcal{G}}} \underline{\mathsf{MP}}^{\mathcal{G}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1)) - \epsilon$.*

Proof. Consider a nonzero-sum mean-payoff game $\mathcal{G}$ and a vertex v in $\mathcal{G}$ and a strategy σ_0 of Player 0. We let $\inf_{\sigma_1 \in \mathsf{BR}^{2\epsilon}_{1,\mathcal{G}}} \underline{\mathsf{MP}}^{\mathcal{G}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1)) = c$, for some $c \in \mathbb{Q}$. Let the supremum of the payoffs that Player 1 gets when Player 0 plays σ_0 be y, where $y \in \mathbb{Q}$, i.e., $\sup\{\underline{\mathsf{MP}}_1(\rho) \mid \rho \in \mathsf{Out}_v(\mathcal{G}, \sigma_0))\} = y$. For all outcomes ρ which are in Player 1's 2ϵ-best response of σ_0, we have that $\underline{\mathsf{MP}}_1(\rho) > y - 2\epsilon$ and $\underline{\mathsf{MP}}_0(\rho) \geqslant c$.

Now, consider a game $\mathcal{H} \in \mathcal{G}^{\pm\epsilon}$ and a Player 0 strategy σ_0 played in $\mathcal{H}$. We can see that the maximum payoff that Player 1 gets when Player 0 plays σ_0 is bounded by $y + \epsilon$ and $y - \epsilon$, i.e., $y - \epsilon < \sup\{\underline{\mathsf{MP}}_1(\rho) \mid \rho \in \mathsf{Out}_v(\mathcal{H}, \sigma_0))\} < y + \epsilon$. We let this value be denoted by $y_{\mathcal{H}}$. We note that if $\sup_{\rho \in \mathsf{Out}_v(\mathcal{H},\sigma_0)}(\underline{\mathsf{MP}}_1(\rho)) = y_{\mathcal{H}}$, then for the corresponding play $\rho_{\mathcal{H}}$ in the game $\mathcal{G}$, the mean-payoff of Player 1 in $\rho_{\mathcal{H}}$ is $\underline{\mathsf{MP}}_1(\rho_{\mathcal{H}}) > y - 2\epsilon$. Thus, in the game $\mathcal{G}$, we note that $\underline{\mathsf{MP}}_0(\rho_{\mathcal{H}}) \geqslant c$ and for the corresponding play in $\mathcal{H}$, we have $\underline{\mathsf{MP}}_0(\rho_{\mathcal{H}}) > c - \epsilon$. Thus, we have $\mathsf{ASV}_{\mathcal{H}}(\sigma_0)(v) > c - \epsilon = \inf_{\sigma_1 \in \mathsf{BR}^{2\epsilon}_{1,\mathcal{G}}} \underline{\mathsf{MP}}^{\mathcal{G}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1)) - \epsilon$. ◀

We note that in the above theorem, we need to consider a strategy that is robust against 2ϵ-best-responses to ensure robustness against ϵ weight perturbations.

ϵ-Adversarial Stackelberg Value. The results above suggest that, in order to obtain some robustness guarantees in nonzero-sum mean-payoff games, we must consider a solution concept that accounts for ϵ-best responses of the adversary. This leads to the following definition: Given an $\epsilon > 0$, we define the adversarial value of Player 0 strategy σ_0 when Player 1 plays ϵ-best-responses as

$$\mathsf{ASV}^{\epsilon}(\sigma_0)(v) = \inf_{\sigma_1 \in \mathsf{BR}^{\epsilon}_1(\sigma_0)} \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1)) \tag{1}$$

and the ϵ-Adversarial Stackelberg value at vertex v is: $\mathsf{ASV}^{\epsilon}(v) = \sup_{\sigma_0 \in \Sigma_0} \mathsf{ASV}^{\epsilon}(\sigma_0)(v)$, and we note that $\mathsf{ASV}(v) = \sup_{\epsilon > 0} \mathsf{ASV}^{\epsilon}(v)$. We can now state a theorem about combined robustness of ASV^{ϵ}.

▶ **Theorem 6** (Combined robustness of ASV^{ϵ}). *For all nonzero-sum mean-payoff games $\mathcal{G}$ with a set V of vertices, for all $\epsilon > 0$, for all $\delta > 0$, for all $\mathcal{H} \in \mathcal{G}^{\pm\delta}$, for all vertices $v \in V$, and for all strategies σ_0, we have that if $\mathsf{ASV}^{2\delta+\epsilon}_{\mathcal{G}}(\sigma_0)(v) > c$, then for all $\mathcal{H} \in \mathcal{G}^{\pm\delta}$, we have that $\inf_{\sigma_1 \in \mathsf{BR}^{\epsilon}_{\mathcal{H}}(\sigma_0)} \underline{\mathsf{MP}}^{\mathcal{H}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1)) > c - \delta$.*

Proof. The proof for Theorem 6 is very similar to the proof of Theorem 5 and involves looking at the set of ϵ-best-responses in the game $\mathcal{H}$ and showing that the corresponding plays lie in the set of $(2\delta + \epsilon)$-best-responses in the game $\mathcal{G}$. This would imply that the corresponding Player 0 mean-payoffs for the ϵ-best-responses of Player 1 in every perturbed game $\mathcal{H} \in \mathcal{G}^{\pm\delta}$ would always be greater than $c - \delta$. Therefore, we can extrapolate that $\mathsf{ASV}^{\epsilon}_{\mathcal{H}}(\sigma_0)(v) > c - \delta$. ◀

In the rest of the paper we study properties of ASV^{ϵ} and solve the following two problems:

- Threshold Problem of ASV^{ϵ}: Given $\mathcal{G}$, $c \in \mathbb{Q}$, an $\epsilon > 0$, and a vertex v, we provide a nondeterministic polynomial time algorithm to decide if $\mathsf{ASV}^{\epsilon}(v) > c$ (see Theorem 8).

- Computation of ASV^ϵ and largest ϵ: Given $\mathcal{G}$, an $\epsilon > 0$, and a vertex v, we provide an exponential time algorithm to compute $\mathsf{ASV}^\epsilon(v)$ (see Theorem 12). We also establish that ASV^ϵ is achievable (see Theorem 16). Then we show, given a fixed threshold c, how to computation of largest ϵ such that $\mathsf{ASV}^\epsilon(v) > c$. Formally, we compute $\sup\{\epsilon > 0 \mid \mathsf{ASV}^\epsilon(v) > c\}$ (See Corollary 15).

4 Threshold problem for the ASV^ϵ

In this section, given $c \in \mathbb{Q}$, and a vertex v in game $\mathcal{G}$, we study the threshold problem which is to determine if $\mathsf{ASV}^\epsilon(v) > c$.

Witnesses for ASV^ϵ. For a game $\mathcal{G}$ and $\epsilon > 0$, we associate with each vertex v in $\mathcal{G}$, a set $\Lambda^\epsilon(v)$ of pairs or real numbers (c, d) such that Player 1 has a strategy that ensures a mean-payoff greater than d - ϵ for himself while restricting the payoff of Player 0 to at most c. Formally, we have:

$$\Lambda^\epsilon(v) = \{(c,d) \in \mathbb{R}^2 \mid v \vDash \ll 1 \gg \underline{\mathsf{MP}}_0 \leqslant c \wedge \underline{\mathsf{MP}}_1 > d - \epsilon\}.$$

A vertex v is $(c,d)^\epsilon$-bad if $(c,d) \in \Lambda^\epsilon(v)$. Let $c' \in \mathbb{R}$. A play π of $\mathcal{G}$ is called a $(c',d)^\epsilon$-witness of $\mathsf{ASV}^\epsilon(v) > c$ if $(\underline{\mathsf{MP}}_0(\pi), \underline{\mathsf{MP}}_1(\pi)) = (c', d)$ where $c' > c$, and π does not contain any $(c,d)^\epsilon$-bad vertex. A play π is called a witness for $\mathsf{ASV}^\epsilon(v) > c$ if it is a $(c',d)^\epsilon$-witness for $\mathsf{ASV}^\epsilon(v) > c$ for some c', d. We now show that polynomial-size witnesses for $\mathsf{ASV}^\epsilon > c$ exist:

▶ **Theorem 7.** *For all mean-payoff games $\mathcal{G}$, for all vertices v in $\mathcal{G}$, for all $\epsilon > 0$, and $c \in \mathbb{Q}$, we have that $\mathsf{ASV}^\epsilon(v) > c$ if and only if there exists a $(c',d)^\epsilon$-witness of $\mathsf{ASV}^\epsilon(v) > c$, where $d \in \mathbb{Q}$. Furthermore, the $(c',d)^\epsilon$-witness can be chosen as a regular witness $\pi = u \cdot v^\omega$, where u and v are finite paths of polynomial size.*

Proof sketch. We consider only the left to right direction here since the other direction of the proof is similar to showing that existence of a witness for $\mathsf{ASV}(v) > c$ implies $\mathsf{ASV}(v) > c$ [1]. We are given that $\mathsf{ASV}^\epsilon(v) > c$. First we show that $\mathsf{ASV}^\epsilon(v) > c$ iff there exists a strategy σ_0 of Player 0 such that $\mathsf{ASV}^\epsilon(\sigma_0)(v) > c$. Thus, there exists a $\delta > 0$, such that $\inf_{\sigma_1 \in \mathsf{BR}_1^\epsilon(\sigma_0)} \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1)) = c' = c + \delta$ Let $d = \sup_{\sigma_1 \in \mathsf{BR}_1^\epsilon(\sigma_0)} \underline{\mathsf{MP}}_1(\mathsf{Out}_v(\sigma_0, \sigma_1))$. We show that for all $\sigma_1 \in \mathsf{BR}_1^\epsilon(\sigma_0)$, we have that $\mathsf{Out}_v(\sigma_0, \sigma_1)$ does not cross a $(c,d)^\epsilon$-bad vertex. We then consider a sequence $(\sigma_i)_{i \in \mathbb{N}}$ of Player 1 strategies such that $\sigma_i \in \mathsf{BR}_1^\epsilon(\sigma_0)$ for all $i \in \mathbb{N}$, and $\lim_{i \to \infty} \underline{\mathsf{MP}}_1(\mathsf{Out}_v(\sigma_0, \sigma_i)) = d$. Let $\pi_i = \mathsf{Out}_v(\sigma_0, \sigma_i)$. W.l.o.g., we can have that all the plays $\mathsf{Out}_v(\sigma_0, \sigma_i)$ end up in the same SCC, say S.

Now using the fact that $\mathsf{F}_{\min}(\mathsf{CH}(\mathbb{C}(S)))$ is a closed set, and using a result from [8, 5] which states that for every pair of points (x, y) in $\mathsf{F}_{\min}(\mathsf{CH}(\mathbb{C}(S)))$, there exists a play π in the SCC S such that $(\underline{\mathsf{MP}}_0(\pi), \underline{\mathsf{MP}}_1(\pi)) = (x, y)$, we can show that there exists a play $\pi^* \in \mathsf{Out}_v(\sigma_0)$ with $(\underline{\mathsf{MP}}_0(\pi^*), \underline{\mathsf{MP}}_1(\pi^*)) = (c^*, d)$ and $c^* \geqslant c'$. That π^* is a $(c^*, d)^\epsilon$-witness now follows since the vertices appearing in π^* are not $(c,d)^\epsilon$-bad. We have thus shown that if $\mathsf{ASV}^\epsilon(v) > c$, then there exists a $(c^*, d)^\epsilon$-witness. Finally, by using the Carathéodory baricenter theorem, we show that two simple cycles, and three acyclic finite plays suffice to construct a regular witness. ◀

The following statement can be obtained by exploiting the existence of finite regular witnesses of polynomial size proved above.

▶ **Theorem 8.** *For all mean-payoff games $\mathcal{G}$, for all vertices v in $\mathcal{G}$, for all $\epsilon > 0$, and for all $c \in \mathbb{Q}$, it can be decided in nondeterministic polynomial time if $\mathsf{ASV}^{\epsilon}(v) > c$, and a pseudopolynomial memory strategy of Player 0 suffices for this threshold. Furthermore, this decision problem is at least as hard as solving zero-sum mean-payoff games.*

As a corollary of Theorem 8, we can deduce that the ϵ-adversarial Stackelberg value achievable using finite memory strategies which defined as :

$$\mathsf{ASV}^{\epsilon}_{\mathsf{FM}}(v) = \sup_{\sigma_0 \in \Sigma_0^{\mathsf{FM}}} \inf_{\sigma_1 \in \mathsf{BR}_1^{\epsilon}(\sigma_0)} \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1))$$

where Σ_0^{FM} refers to the set of all finite memory strategies of Player 0, is equal to ASV^{ϵ}:

▶ **Corollary 9.** *For all games $\mathcal{G}$, for all vertices v in $\mathcal{G}$, and for all $\epsilon > 0$, we have that $\mathsf{ASV}^{\epsilon}_{\mathsf{FM}}(v) = \mathsf{ASV}^{\epsilon}(v)$.*

This corollary is important from a practical point of view as it implies that the ASV^{ϵ} value can be approached to any precision with a finite memory strategy. Nevertheless, we show in Theorem 16 that infinite memory is necessary to achieve the exact ASV^{ϵ}.

Memoryless strategies of Player 0. We now establish that the threshold problem is NP-complete when Player 0 is restricted to play *memoryless* strategies. First we define

$$\mathsf{ASV}^{\epsilon}_{\mathsf{ML}}(v) = \sup_{\sigma_0 \in \Sigma_0^{\mathsf{ML}}} \inf_{\sigma_1 \in \mathsf{BR}_1^{\epsilon}(\sigma_0)} \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0, \sigma_1))$$

where Σ_0^{ML} is the set of all memoryless strategies of Player 0.

▶ **Theorem 10.** *For all mean-payoff games $\mathcal{G}$, for all vertices v in $\mathcal{G}$, for all $\epsilon > 0$, and for all rationals c, the problem of deciding if $\mathsf{ASV}^{\epsilon}_{\mathsf{ML}}(v) > c$ is* NP-Complete.

The proof of hardness is a reduction from the partition problem while easiness is straightforwardly obtained by techniques used in the proof of Theorem 8.

5 Computation of the ASV^{ϵ} and the largest ϵ possible

Here, we express the ASV^{ϵ} as a formula in the theory of reals by adapting a method provided in [8] for ASV. We then provide a new EXPTime algorithm to compute the ASV^{ϵ} based on construction of linear programs (LPs) which in turn is applicable to ASV as well.

Extended mean-payoff game. Given a mean-payoff game $\mathcal{G}$ with a set V of vertices in its arena, we construct an extended mean-payoff game $\mathcal{G}^{\mathsf{ext}}$, whose arena consists of vertices $V^{\mathsf{ext}} = V \times 2^V$. With a history h in $\mathcal{G}$, we associate a vertex in $\mathcal{G}^{\mathsf{ext}}$ which is a pair (v, P), where $v = \mathsf{last}(h)$ and P is the set of the vertices traversed along h. The set E^{ext} of edges, and the weight functions w_i^{ext} for $i \in \{0, 1\}$ are defined as $E^{\mathsf{ext}} = \{((v, P), (v', P')) \mid (v, v') \in E, P' = P \cup \{v'\}\}$ and $w_i^{\mathsf{ext}}((v, P), (v', P')) = w_i(v, v')$ respectively. There exists a bijection between the plays π in $\mathcal{G}$ and the plays π^{ext} in $\mathcal{G}^{\mathsf{ext}}$. Note that the second component of the vertices of the play π^{ext} stabilises into a set of vertices of $\mathcal{G}$ which we denote by $V^*(\pi^{\mathsf{ext}})$.

We characterize $\mathsf{ASV}^{\epsilon}(v)$ with the notion of witness introduced earlier and the decomposition of $\mathcal{G}^{\mathsf{ext}}$ into SCCs. For a vertex v in V, let $\mathsf{SCC}^{\mathsf{ext}}(v)$ be the set of strongly-connected components in $\mathcal{G}^{\mathsf{ext}}$ which are reachable from $(v, \{v\})$.

▶ **Lemma 11.** *For all mean-payoff games $\mathcal{G}$ and for all vertices v in $\mathcal{G}$, we have*

$$\mathsf{ASV}^{\epsilon}(v) = \max_{S \in \mathsf{SCC}^{\mathsf{ext}}(v)} \sup\{c \in \mathbb{R} \mid \exists \pi^{\mathsf{ext}} : \pi^{\mathsf{ext}} \textit{ is a witness for } \mathsf{ASV}^{\epsilon}(v, \{v\}) > c \textit{ and } V^*(\pi^{\mathsf{ext}}) = S\}.$$

By definition of $\mathcal{G}^{\mathsf{ext}}$, for every SCC S of $\mathcal{G}^{\mathsf{ext}}$, there exists a set $V^*(S)$ of vertices of $\mathcal{G}$ such that every vertex of S is of the form $(v', V^*(S))$, where v' is a vertex in $\mathcal{G}$. Now, we define $\Lambda^{\mathsf{ext}}_S = \bigcup_{v \in V^*(S)} \Lambda^{\epsilon}(v)$ as the set of (c, d) such that Player 1 can ensure $v \vDash \ll 1 \gg \underline{\mathsf{MP}}_0 \leqslant c \wedge \underline{\mathsf{MP}}_1 > d - \epsilon$ from some vertex $v \in S$. The set Λ^{ext}_S can be represented by a formula $\Psi^{\epsilon}_S(x, y)$ in the first order theory of reals with addition, $\langle \mathbb{R}, +, < \rangle$, with two free variables. We refer the reader to [1] for a formal statement and a proof of this. We can now state the following theorem about the computability of $\mathsf{ASV}^{\epsilon}(v)$:

▶ **Theorem 12.** *For all mean-payoff games $\mathcal{G}$, for all vertices v in $\mathcal{G}$ and for all $\epsilon > 0$, the $\mathsf{ASV}^{\epsilon}(v)$ can be effectively expressed by a formula in $\langle \mathbb{R}, +, < \rangle$, and can be computed from this formula. Furthermore, the formula can be effectively transformed into exponentially many linear programs which establish membership in* EXPTime.

Proof sketch. Using Lemma 11, we have that for every S in $\mathsf{SCC}^{\mathsf{ext}}(v)$, a value of c such that $\mathsf{ASV}^{\epsilon}(v) > c$ can be encoded by the formula $\rho^S_v(c) \equiv \exists x, y \cdot x > c \wedge \Phi_S(x, y) \wedge \neg \Psi^{\epsilon}_S(c, y)$ where the formula $\Phi_S(x, y)$ expresses the symbolic encoding of the pair of values (x, y) which represents the mean-payoff values of some play in S, and the formula $\neg \Psi^{\epsilon}_S(c, y)$ expresses that the play does not cross a $(c, y)^{\epsilon}$-bad vertex. We then construct a formula $\rho^S_{\max,v}(z)$ which is satisfied by a value that is the supremum over the set of values c such that c satisfies the formula ρ^S_v. From the formula $\rho^S_{\max,v}$, we can compute the $\mathsf{ASV}^{\epsilon}(v)$ by quantifier elimination, and by finding the maximum across all the SCCs S in $\mathsf{SCC}^{\mathsf{ext}}(v)$.

For the EXPTime algorithm, first note that for each SCC S in $\mathcal{G}^{\mathsf{ext}}$, the set satisfying the formula $\Phi_S(x, y)$, which is the symbolic encoding of $\mathsf{F}_{\min}(\mathsf{CH}(\mathbb{C}(S)))$, can be expressed as a set of *exponentially* many inequalities [5]. Also the formula $\Psi^{\epsilon}_S(x, y)$ can be expressed using exponentially many LPs. We refer the interested reader to [1] for more details. It follows that the formula $\rho^S_v(c)$ can be expressed with exponentially many LPs. In each LP, the objective is to maximize c. The algorithm runs in EXPTime since there can be exponentially many SCCs. ◀

▶ **Example 13.** We illustrate the computation of ASV^{ϵ} with an example. Consider the mean-payoff game $\mathcal{G}$ depicted in Figure 4 and its extension $\mathcal{G}^{\mathsf{ext}}$ as shown in Figure 5.

Note that in $\mathcal{G}^{\mathsf{ext}}$ there exist three SCCs which are $S_1 = \{v'_{0^2}, v'_1\}, S_2 = \{v'_{2^1}\}$, and $S_3 = \{v'_{2^2}\}$. The SCCs S_2 and S_3 are similar, and thus $\rho^{S_2}_{\max,v_0}(z)$ and $\rho^{S_3}_{\max,v_0}(z)$ would be equivalent. We start with SCC S_1 that contains two cycles $v'_1 \to v'_1$ and $v'_{0^2} \to v'_1, v'_1 \to v'_{0^2}$, and SCC S_2 contains one cycle $v'_{2^1} \to v'_{2^1}$. Since S_3 is similar to S_2, we consider only S_2 in our example. Thus, the set $\mathsf{F}_{\min}(\mathsf{CH}(\mathbb{C}(S_1)))$ is represented by the Cartesian points within the triangle represented by $(0, 2), (1, 1)$ and $(0, 1)$ [5] and $\mathsf{F}_{\min}(\mathsf{CH}(\mathbb{C}(S_2))) = \{(0, 1)\}$. Thus, we get that $\Phi_{S_1}(x, y) \equiv (x \geqslant 0 \wedge x \leqslant 1) \wedge (y \geqslant 1 \wedge y \leqslant 2) \wedge (x + y) \leqslant 2$ and $\Phi_{S_2}(x, y) \equiv x = 0 \wedge y = 1$. Now, we calculate $\Lambda^{\epsilon}(v'_{0^2})$, $\Lambda^{\epsilon}(v'_{2^1})$ and $\Lambda^{\epsilon}(v'_1)$ for some value of ϵ less tan 1. We note that the vertex v'_1 is not $(0, 2 + \epsilon - \delta)^{\epsilon}$-bad, for all $0 < \delta < 1$, as Player 0 can always choose the edge (v'_1, v'_{0^2}) from v'_1, thus giving Player 1 a mean-payoff of 1. Additionally, the vertex v'_{0^2} is both $(0, 1 + \epsilon - \delta)^{\epsilon}$-bad, for all $\delta > 0$, since Player 1 can choose the edge (v'_{0^2}, v'_{2^2}) from v'_{0^2}, and

[5] Note that the coordinate $(0, 1)$ is obtained as the pointwise minimum over the two coordinates separately.

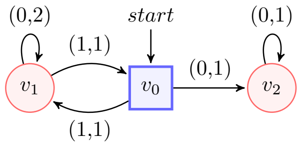


Figure 4 Example to calculate $\mathsf{ASV}^\epsilon(v)$.

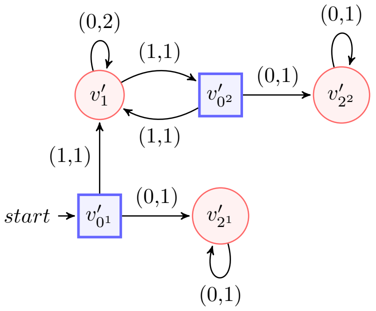


Figure 5 Extended Mean-Payoff Game where $v'_{0^1} = (v_0, \{v_0\})$, $v'_{0^2} = (v_0, \{v_0, v_1\})$, $v'_1 = (v_1, \{v_0, v_1\})$, $v'_{2^1} = (v_1, \{v_0, v_2\})$, and $v'_{2^2} = (v_2, \{v_0, v_1, v_2\})$.

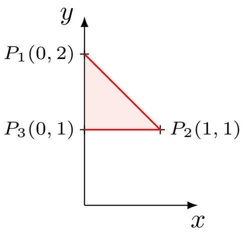


Figure 6 The red triangle represents the set of points in Φ_{S_1}.

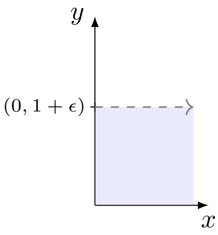


Figure 7 The blue region under and excluding the line $y = (1-\epsilon)$ represents the set of points in $\Psi^\epsilon_{S_1}$ and $\Psi^\epsilon_{S_2}$.

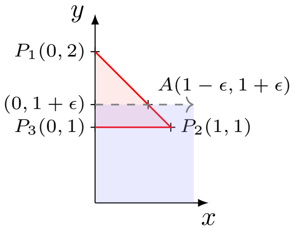


Figure 8 The formula $\rho^{S_1}(c)$ is represented by the points in Φ_{S_1} and not in $\Psi^\epsilon_{S_1}$, i.e., the points in the triangle which are not strictly below the line $y = (1-\epsilon)$. Here, the max c value is represented by point A.

$(1, 1+\epsilon-\delta)^\epsilon$-bad, for all $\delta > 0$, since Player 1 can choose the edge (v'_{0^2}, v'_1) from v'_{0^2}. Thus, we get that $\Lambda^\epsilon(v'_1) = \Lambda^\epsilon(v'_{0^2}) = \{(c,d) \mid (c \geqslant 1 \wedge d < 1+\epsilon)\} \bigcup \{(c,d) \mid (c \geqslant 0 \wedge d < 1+\epsilon)\}$ which is the same as $\{(c,d) \mid (c \geqslant 0 \wedge d < 1+\epsilon)\}$, and $\Lambda^\epsilon(v'_{2^1}) = \{(c,d) \mid (c \geqslant 0 \wedge d < 1+\epsilon)\}$. Therefore, we have that $\Lambda^{\mathsf{ext}}_{S_1} = \Lambda^{\mathsf{ext}}_{S_2} = \{(c,d) \mid (c \geqslant 0 \wedge d < 1+\epsilon)\}$. Hence, we get that $\Psi^\epsilon_{S_1}(x,y) = \Psi^\epsilon_{S_2}(x,y) \equiv (x \geqslant 0 \wedge y < 1+\epsilon)$ as shown in Figure 7. From Figure 8, the formula $\rho^{S_1}(c)$ holds true for values of c less than $(1-\epsilon)$ and the formula $\rho^{S_2}(c)$ holds for values of c less than 0. Hence, by assigning $(1-\epsilon)$ to x, and $(1+\epsilon)$ to y, we get that $\rho^{S_1}_{\max, v_0}(z)$ holds true for $z = (1-\epsilon)$. Additionally, by assigning 0 to x, and 1 to y, we get that $\rho^{S_2}_{\max, v_0}(z)$ holds true for $z = 0$. It follows that $\mathsf{ASV}^\epsilon(v_0) = 1-\epsilon$ for $\epsilon < 1$ as it is the maximum of the values over all the SCCs. ◀

We now illustrate the LP formulation for $\rho^S_v(c)$ for each SCC S with the following example, and provide details for computing $\mathsf{ASV}^\epsilon(v_0)$.

▶ **Example 14.** We previously showed that the $\mathsf{ASV}^\epsilon(v_0)$ can be computed by quantifier elimination of a formula in the theory of reals with addition. Now, we compute the $\mathsf{ASV}^\epsilon(v_0)$ by solving a set of linear programs for every SCC in $\mathcal{G}^{\mathsf{ext}}$. We recall that there are three SCCs S_1, S_2 and S_3 in $\mathcal{G}^{\mathsf{ext}}$. From a result in [5], we have that $\mathsf{F}_{\min}(\mathsf{CH}(\mathbb{C}(S_i)))$ for $i \in \{1,2,3\}$ can be defined using a set of linear inequalities. Now recall that $\mathsf{F}_{\min}(\mathsf{CH}(\mathbb{C}(S_2)) = \mathsf{F}_{\min}(\mathsf{CH}(\mathbb{C}(S_3))) = \{(0,1)\}$, and $\mathsf{F}_{\min}(\mathsf{CH}(\mathbb{C}(S_1)))$ is represented by the set of points enclosed by the triangle formed by connecting the points $(0,1), (1,1)$ and $(0,2)$ as shown in Figure 6, and $\Lambda^\epsilon(v'_{0^2}) = \Lambda^\epsilon(v'_{2^1}) = \Lambda^\epsilon(v'_{2^2}) = \Lambda^\epsilon(v'_1) = \{(c,y) \,|\, c \geqslant 0 \wedge y < 1+\epsilon\}$. Now, we consider the

SCC S_1, and the formula $\neg\Psi^{\epsilon}_{S_1}$. We start this by finding the complement of $\Lambda^{\epsilon}(v'_{0^2})$ and $\Lambda^{\epsilon}(v'_1)$, that is, $\overline{\Lambda}^{\epsilon}(v'_{0^2}) = \overline{\Lambda}^{\epsilon}(v'_1) = \mathbb{R}\times\mathbb{R} - \Lambda^{\epsilon}(v'_{0^2}) = \mathbb{R}\times\mathbb{R} - \Lambda^{\epsilon}(v'_1) = \{(c,y) \mid c < 0 \vee y \geqslant 1+\epsilon\}$. Now, we get that $\neg\Psi^{\epsilon}_{S_1} = \overline{\Lambda}^{\epsilon}(v'_{0^2}) \bigcap \overline{\Lambda}^{\epsilon}(v'_1) = \{(c,y) \mid c < 0 \vee y \geqslant 1+\epsilon\}$.

Similarly for the SCC S_2 and SCC S_3, we calculate the complement of $\Lambda^{\epsilon}(v'_{2^1})$ and $\Lambda^{\epsilon}(v'_{2^2})$, that is, $\overline{\Lambda}^{\epsilon}(v'_{2^2}) = \overline{\Lambda}^{\epsilon}(v'_{2^1}) = \mathbb{R}\times\mathbb{R} - \Lambda^{\epsilon}(v'_{2^1}) = \mathbb{R}\times\mathbb{R} - \Lambda^{\epsilon}(v'_{2^2}) = \{(c,y) \mid c < 0 \vee y \geqslant 1+\epsilon\}$ and obtain $\neg\Psi^{\epsilon}_{S_2} = \overline{\Lambda}^{\epsilon}(v'_{2^1}) = \{(c,y) \mid c < 0 \vee y \geqslant 1+\epsilon\}$ and $\neg\Psi^{\epsilon}_{S_3} = \overline{\Lambda}^{\epsilon}(v'_{2^2}) = \{(c,y) \mid c < 0 \vee y \geqslant 1+\epsilon\}$. Note that the formulaes $\Phi_{S_2}(x,y)$ and $\Phi_{S_3}(x,y)$ are represented by the set of linear inequations $x = 0 \wedge y = 1$ and the formula $\Phi_{S_1}(x,y)$ is represented by the set of linear inequations $y \geqslant 1 \wedge y \leqslant 2 \wedge x \leqslant 1 \wedge (x+y) \leqslant 2$. Now the formula $\rho^{S_1}_{v_0}(c)$ can be expressed using a set of linear equations and inequalities as follows: $x > c \wedge y \geqslant 1 \wedge y \leqslant 2 \wedge x \leqslant 1 \wedge (x+y) \leqslant 2 \wedge (c < 0 \vee y \geqslant 1+\epsilon)$ and the formula $\rho^{S_2}_{v_0}(c)$ can be expressed using a set of linear equations and inequalities as follows: $x > c \wedge x = 0 \wedge y = 1 \wedge (c < 0 \vee y \geqslant 1+\epsilon)$. We maximise the value of c in the formula $\rho^{S_1}_{v_0}(c)$ to get the following two linear programs: *maximise* c *in* $(x > c \wedge y \geqslant 1 \wedge y \leqslant 2 \wedge x \leqslant 1 \wedge (x+y) \leqslant 2 \wedge c < 0)$ which gives a solution $\{0\}$ and *maximise* c *in* $(x > c \wedge y \geqslant 1 \wedge y \leqslant 2 \wedge x \leqslant 1 \wedge (x+y) \leqslant 2 \wedge y \geqslant (1+\epsilon))$ which gives us a solution $\{(1-\epsilon)\}$. Similarly, maximising c in the formulaes $\rho^{S_2}_{v_0}(c)$ and $\rho^{S_3}_{v_0}(c)$ would give us the following two linear programs: *maximise* c *in* $(x > c \wedge x = 0 \wedge y = 1 \wedge c < 0)$ which gives a solution $\{0\}$ and *maximise* c *in* $(x > c \wedge x = 0 \wedge y = 1 \wedge y \geqslant (1+\epsilon))$ which gives us a solution $\{0\}$. Thus, we conclude that $\mathsf{ASV}^{\epsilon}(v_0) = 1-\epsilon$ which is the maximum value amongst all the SCCs. Note that in an LP, the strict inequalities are replaced with non-strict inequalities, and computing the supremum in the objective function is replaced by maximizing the objective function.

Again, for every SCC S and for every LP corresponding to that of S, we fix a value of c and change the objective function to *maximise* ϵ from *maximise* c in order to obtain the maximum value of ϵ that allows $\mathsf{ASV}^{\epsilon}(v_0) > c$. For example, consider the LP $(x > c \wedge y \geqslant 1 \wedge y \leqslant 2 \wedge x \leqslant 1 \wedge (x+y) \leqslant 2 \wedge y \geqslant (1+\epsilon))$ in SCC S_1 and fix a value of c, and then maximize the value of ϵ. Doing this over all linear programs in an SCC, and over all SCCs, reachable from v_0 for a fixed c gives us the supremum value of ϵ such that we have $\mathsf{ASV}^{\epsilon}(v_0) > c$. ◀

On the other hand, we note that in every SCC S, the value c is a function of ϵ, for illustration, in the example above, $\rho^{S_1}(c)$ holds true for values of c less than $1-\epsilon$. Thus if we fix a value of c, we can find the supremum over ϵ which allows $\mathsf{ASV}^{\epsilon}(v) > c$ in S. Again, taking the maximum over all SCCs reachable from $(v, \{v\})$ gives us the largest ϵ possible so that we have $\mathsf{ASV}^{\epsilon}(v) > c$. We state the following corollary.

▶ **Corollary 15.** *For all mean-payoff games $\mathcal{G}$, for all vertices v in $\mathcal{G}$, and for all $c \in Q$, we can compute in* EXPTime *the maximum possible value of ϵ such that* $\mathsf{ASV}^{\epsilon}(v) > c$.

6 Additional Properties of ASV^{ϵ}

In this section, we first show that the ASV^{ϵ} is *achievable*, i.e., there exists a Player 0 strategy that achieves the ASV^{ϵ}. Then we study the memory requirement in strategies of Player 0 for achieving the ASV^{ϵ}, as well as the memory requirement by Player 1 for playing the ϵ-best-responses.

Achievability of the ASV^{ϵ}. We formally define achievability as follows. Given $\epsilon > 0$, we say that $\mathsf{ASV}^{\epsilon}(v) = c$ is achievable from a vertex v, if there exists a strategy σ_0 for Player 0 such that $\forall \sigma_1 \in \mathsf{BR}^{\epsilon}_1(\sigma_0) : \underline{\mathsf{MP}}_0(\mathsf{Out}_v(\sigma_0,\sigma_1)) \geqslant c$. We note that this result is in contrast to the case for ASV as shown in [8].

▶ **Theorem 16.** *For all mean-payoff games $\mathcal{G}$, for all vertices v in $\mathcal{G}$, and for all $\epsilon > 0$, we have that the $\mathsf{ASV}^{\epsilon}(v)$ is achievable.*

The rest of this section is devoted to proving Theorem 16. We start by defining the notion of a witness for $\mathsf{ASV}^{\epsilon}(\sigma_0)(v)$ for a strategy σ_0 of Player 0.

Witness for $\mathsf{ASV}^{\epsilon}(\sigma_0)(v)$. Given a mean-payoff game $\mathcal{G}$, a vertex v in $\mathcal{G}$, and an $\epsilon > 0$, we say that a play π is a witness for $\mathsf{ASV}^{\epsilon}(\sigma_0)(v) > c$ for a strategy σ_0 of Player 0 if (i) $\pi \in \mathsf{Out}_v(\sigma_0)$, and (ii) π is a witness for $\mathsf{ASV}^{\epsilon}(v) > c$ when Player 0 uses strategy σ_0 where the strategy σ_0 is defined as follows:

1. σ_0 follows π if Player 1 does not deviate from π.
2. If Player 1 deviates π, then for each vertex $v \in \pi$, we have that σ_0 consists of a memoryless strategy that establishes $v \nvDash \ll 1 \gg \underline{\mathsf{MP}}_0 \leqslant c \wedge \underline{\mathsf{MP}}_1 > d - \epsilon$, where $d = \underline{\mathsf{MP}}_1(\pi)$. The existence of such a memoryless strategy of Player 0 has been established in Section 4.

Assume that the $\mathsf{ASV}^{\epsilon}(v)$ cannot be achieved by a finite memory strategy. We show that for such cases, it can indeed be achieved by an infinite memory strategy.

Let $\mathsf{ASV}^{\epsilon}(v) = c$. For every $c' < c$, from Theorem 8, there exists a finite memory strategy σ_0 such that $\mathsf{ASV}^{\epsilon}(\sigma_0)(v) > c'$, and recall from Theorem 7 that there exists a corresponding regular witness. First we state the following proposition.

▶ **Proposition 17.** *There exists a sequence of increasing real numbers, $c_1 < c_2 < c_3 < \ldots < c$, such that the sequence converges to c, and a set of finite memory strategies $\sigma_0^1, \sigma_0^2, \sigma_0^3, \ldots$ of Player 0 such that for each c_i, we have $\mathsf{ASV}^{\epsilon}(\sigma_0^i)(v) > c_i$, and there exists a play π^i that is a witness for $\mathsf{ASV}^{\epsilon}(\sigma_0^i)(v) > c_i$, where $\pi^i = \pi_1(l_1^{\alpha \cdot k_i} \cdot \pi_2 \cdot l_2^{\beta \cdot k_i} \cdot \pi_3)^{\omega}$, and π_1, π_2 and π_3 are simple finite plays, and l_1, l_2 are simple cycles in the arena of the game $\mathcal{G}$.*

These witnesses or plays in the sequence are regular, and they differ from each other only in the value of k_i that they use.

To show that $\lim\limits_{i \to \infty} \mathsf{ASV}^{\epsilon}(\sigma_0^i)(v) = c$, we construct a play π^* that starts from v, follows π^1 until the mean-payoff of Player 0 over the prefix becomes greater than c_1. Then for $i \in \{2, 3, \ldots\}$, starting from $\mathsf{first}(l_1)$, it follows π^i, excluding the initial simple finite play π_1, until the mean-payoff of the prefix of π^i becomes greater than c_i. Then the play π^* follows the prefix of the play π^{i+1}, excluding the initial finite play π_1, and so on. Clearly, we have that $\underline{\mathsf{MP}}_1(\pi^*) = c$. We let $\underline{\mathsf{MP}}_1(\pi^*) = d = \alpha \cdot \mathsf{MP}_1(l_1) + \beta \cdot \mathsf{MP}_1(l_2)$.

For the sequence of plays $(\pi^i)_{i \in \mathbb{N}^+}$ which are witnesses for $(\mathsf{ASV}^{\epsilon}(\sigma_0^i)(v) > c_i)_{i \in \mathbb{N}^+}$ for the strategies $(\sigma_0^i)_{i \in \mathbb{N}^+}$, we let $\underline{\mathsf{MP}}_1(\pi_i) = d_i$. We state the following proposition.

▶ **Proposition 18.** *The sequence $(d_i)_{i \in \mathbb{N}^+}$ is monotonic, and it converges to d in the limit.*

The above two propositions establish the existence of an infinite sequence of regular witnesses $\mathsf{ASV}^{\epsilon}(\sigma_0^i)(v) > c_i$ for a sequence of increasing numbers $c_1 < c_2 < \ldots < c$, such that the mean-payoffs of the witnesses are monotonic and at the limit, the mean-payoffs of the witnesses converge to c and d for Player 0 and Player 1 respectively. These observations show the existence of a witness π^* which gives Player 0 a mean-payoff value at least c and Player 1 a mean-payoff value equal to d. Assuming that Player 0 has a corresponding strategy σ_0, we show that Player 1 does not have an ϵ-best response to σ_0 that gives Player 0 a payoff less than c. Now, we have the ingredients to prove Theorem 16.

Proof sketch of Theorem 16. We consider a sequence of increasing numbers $c_1 < c_2 < c_3 < \ldots < c$ such that for every $i \in \mathbb{N}^+$, by Theorem 8, we consider a finite memory strategy σ_0^i of Player 0 that ensures $\mathsf{ASV}^{\epsilon}(\sigma_0^i)(v) > c_i$.

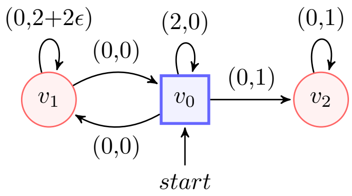


Figure 9 Finite memory strategy of Player 0 may not achieve $\mathsf{ASV}^{\epsilon}(v_0)$. Also, no finite memory ϵ-best response exists for Player 1 for the strategy σ_0 of Player 0.

If the ASV^{ϵ} is not achievable, then there exists a strategy of Player 1 to enforce some play π' such that $\underline{\mathsf{MP}}_0(\pi') = c' < c$ and $\underline{\mathsf{MP}}_1(\pi') = d' > d - \epsilon$. Now, we use the monotonicity of the sequence $(d_i)_{i \in \mathbb{N}^+}$ established in Proposition 18 to show a contradiction. Since the sequence $(d_i)_{i \in \mathbb{N}^+}$ is monotonic, there can be two cases:

1. The sequence $(d_i)_{i \in \mathbb{N}^+}$ is monotonically non-decreasing.
2. The sequence $(d_i)_{i \in \mathbb{N}^+}$ is monotonically decreasing.

For each of these cases, we reach a contradiction if we assume that $\mathsf{ASV}^{\epsilon}(v)$ is not achievable, i.e. Player 1 deviates from π^* to enforce the play π' where $\underline{\mathsf{MP}}_0(\pi') = c' < c$ and $\underline{\mathsf{MP}}_1(\pi') = d'$. ◀

Memory requirements of the players' strategies. First we show that there exists a mean-payoff game $\mathcal{G}$ in which Player 0 needs an infinite memory strategy to achieve the ASV^{ϵ}.

▶ **Theorem 19.** *There exist a mean-payoff game $\mathcal{G}$, a vertex v in $\mathcal{G}$, and an $\epsilon > 0$ such that Player 0 needs an infinite memory strategy to achieve the $\mathsf{ASV}^{\epsilon}(v)$.*

Proof sketch. Consider the example in Figure 9. We show that in this example the $\mathsf{ASV}^{\epsilon}(v_0) = 1$, and that this value can only be achieved using an infinite memory strategy. Assume a strategy σ_0 for Player 0 such that the game is played in rounds. In round k: (i) if Player 1 plays $v_0 \to v_0$ repeatedly at least k times before playing $v_0 \to v_1$, then from v_1, play $v_1 \to v_1$ repeatedly k times and then play $v_1 \to v_0$ and move to round $k+1$; (ii) else, if Player 1 plays $v_0 \to v_0$ less than k times before playing $v_0 \to v_1$, then from v_1 , play $v_1 \to v_0$. Note that σ_0 is an infinite memory strategy. The best-response for Player 1 to strategy σ_0 would be to choose k sequentially as $k = 1, 2, 3, \dots$, to get a play $\pi = ((v_0)^i (v_1)^i)_{i \in \mathbb{N}}$. We have that $\underline{\mathsf{MP}}_1(\pi) = 1 + \epsilon$ and $\underline{\mathsf{MP}}_0(\pi) = 1$. Player 1 can only sacrifice an amount that is less than ϵ to minimize the mean-payoff of Player 0, and thus he would not play $v_0 \to v_2$. We can show that $\mathsf{ASV}^{\epsilon}(\sigma_0)(v_0) = \mathsf{ASV}^{\epsilon}(v_0)$, and that no finite memory strategy of Player 0 can achieve an $\mathsf{ASV}^{\epsilon}(v_0)$ of 1. ◀

There also exist mean-payoff games in which a finite memory (but not memoryless) strategy for Player 0 can achieve the ASV^{ϵ}.

Further, we show that there exist games such that for a strategy σ_0 of Player 0, and an $\epsilon > 0$, there does not exist any finite memory best-response of Player 1 to the strategy σ_0.

▶ **Theorem 20.** *There exist a mean-payoff game $\mathcal{G}$, an $\epsilon > 0$, and a Player 0 strategy σ_0 in $\mathcal{G}$ such that every Player 1 strategy $\sigma_1 \in \mathsf{BR}_1^{\epsilon}(\sigma_0)$ is an infinite memory strategy.*

References

1 Mrudula Balachander, Shibashis Guha, and Jean-François Raskin. Fragility and robustness in mean-payoff adversarial stackelberg games. *CoRR*, abs/2007.07209, 2020. `arXiv:2007.07209`.

2 Roderick Bloem, Krishnendu Chatterjee, Karin Greimel, Thomas A. Henzinger, Georg Hofferek, Barbara Jobstmann, Bettina Könighofer, and Robert Könighofer. Synthesizing robust systems. *Acta Informatica*, 51(3-4):193–220, 2014.

3 Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A. Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas. Non-zero sum games for reactive synthesis. In *Language and Automata Theory and Applications – 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings*, pages 3–23, 2016.

4 Véronique Bruyère, Noémie Meunier, and Jean-François Raskin. Secure equilibria in weighted games. In *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14–18, 2014*, pages 26:1–26:26. ACM, 2014.

5 Krishnendu Chatterjee, Laurent Doyen, Herbert Edelsbrunner, Thomas A. Henzinger, and Philippe Rannou. Mean-payoff automaton expressions. In *CONCUR 2010 – Concurrency Theory, 21th International Conference, Paris, France, August 31-September 3, 2010. Proceedings*, pages 269–283, 2010.

6 Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdzinski. Games with secure equilibria. *Theor. Comput. Sci.*, 365(1-2):67–82, 2006.

7 Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The complexity of rational synthesis. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 121:1–121:15, 2016.

8 Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The adversarial stackelberg value in quantitative games. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, pages 127:1–127:18, 2020.

9 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, pages 190–204, 2010.

10 Anshul Gupta and Sven Schewe. Quantitative verification in rational environments. In *21st International Symposium on Temporal Representation and Reasoning, TIME 2014, Verona, Italy, September 8-10, 2014*, pages 123–131, 2014.

11 Anshul Gupta and Sven Schewe. Buying optimal payoffs in bi-matrix games. *Games*, 9(3):40, 2018.

12 Anshul Gupta, Sven Schewe, Ashutosh Trivedi, Maram Sai Krishna Deepak, and Bharath Kumar Padarthi. Incentive stackelberg mean-payoff games. In *Software Engineering and Formal Methods – 14th International Conference, SEFM 2016, Held as Part of STAF 2016, Vienna, Austria, July 4-8, 2016, Proceedings*, pages 304–320, 2016.

13 Anshul Gupta, Sven Schewe, and Dominik Wojtczak. Making the best of limited memory in multi-player discounted sum games. In *Proceedings Sixth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2015, Genoa, Italy, 21-22nd September 2015*, pages 16–30, 2015.

14 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016.

15 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190, 1989.

16 Heinrich Freiherr von Stackelberg. *Marktform und Gleichgewicht.* Wien und Berlin, J. Springer, 1934.

Continuous Positional Payoffs

Alexander Kozachinskiy
Department of Computer Science, University of Warwick, Coventry, UK

Abstract

What payoffs are positionally determined for deterministic two-player antagonistic games on finite directed graphs? In this paper we study this question for payoffs that are continuous. The main reason why continuous positionally determined payoffs are interesting is that they include the multi-discounted payoffs.

We show that for continuous payoffs positional determinacy is equivalent to a simple property called prefix-monotonicity. We provide three proofs of it, using three major techniques of establishing positional determinacy – inductive technique, fixed point technique and strategy improvement technique. A combination of these approaches provides us with better understanding of the structure of continuous positionally determined payoffs as well as with some algorithmic results.

2012 ACM Subject Classification Theory of computation → Logic

Keywords and phrases Games on graphs, positional strategies, continuous payoffs

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.10

Related Version *Full Version*: arXiv:2012.09047 [19]

Funding Supported by the EPSRC grant EP/P020992/1 (Solving Parity Games in Theory and Practice).

1 Introduction

We study games of the following kind. A game takes place on a finite directed graph. There is a token, initially located in one of the nodes. Before each turn there is exactly one node containing the token. In each turn one of the two antagonistic players called Max and Min chooses an edge starting in a node containing the token. As a result the token moves to the endpoint of this edge, and then the next turn starts. To determine who makes a move in a turn we are given in advance a partition of the nodes into two sets. If the token is in a node from the first set, then Max makes a move, otherwise Min.

Players make infinitely many moves, and this yields an infinite trajectory of the token. Technically, we assume that each node of the graph has at least one out-going edge so that there is always at least one available move. To introduce competitiveness, we should somehow compare the trajectories of the token with each other. For that we first fix some finite set A and label the edges of the game graph by elements of A. We also fix a *payoff* φ which is a function from the set of infinite sequences of elements of A to $\mathbb{R}$. Each possible infinite trajectory of the token is then mapped to a real number called *the reward* of this trajectory as follows: we form an infinite sequence of elements of A by taking the labels of edges along the trajectory, and apply φ to this sequence. The larger the reward is the more Max is happy; on the contrary, Min wants to minimize the reward.

For both of the players we are interested in indicating *an optimal strategy*, i.e., an optimal instruction of how to play in all possible developments of the games. To point out among all the strategies the optimal ones we first introduce a notion of a *value of a strategy*. The value of a Max's strategy σ is the infimum of the payoff over all infinite trajectories, consistent with the strategy. The reward of a play against σ cannot be smaller than its value, but can be arbitrarily close to it. Now, a strategy of Max is called optimal if its value is maximal over all Max's strategies. Similarly, the value of a Min's strategy is the supremum of the payoff over all infinite trajectories, consistent with this Min's strategy. Min's strategies minimizing the value are called optimal.

Observe that the value of any Min's strategy is at least as large as the value of any Max's strategy. A pair (σ, τ) of a Max's strategy σ and a Min's strategy τ is called an *equilibrium* if the value of σ *equals* the value of τ. Both strategies appearing in an equilibrium must be optimal – one proves the optimality of the other. In this paper we only study the so-called *determined* payoffs – payoffs for which all games on finite directed graphs with this payoff have an equilibrium.

For general determined payoffs an optimal strategy might be rather complicated (since the game is infinite, it might even have no finite description). For what determined payoffs both players always have a "simple" optimal strategy? A word "simple" can be understood in different ways [2], and this leads to different classes of determined payoffs. Among these classes we study one for which "simple" is understood in, perhaps, the strongest sense possible. Namely, we study a class of *positionally determined* payoffs.

For a positionally determined payoff all game graphs must have a pair of *positional* strategies which is an equilibrium no matter in which node the game starts. Now, a positional strategy is a strategy which totally ignores the previous trajectory of the token[1] and only looks at its current location. Formally, a positional strategy of Max maps each Max's node to an edge which starts in this node (i.e., to a single edge which Max will use whenever this node contains the token). Min's positional strategies are defined similarly.

A lot of works are devoted to concrete positionally determined payoffs that are of particular interest in other areas of computer science. Classical examples of such payoffs are parity payoffs, mean payoffs and (multi-)discounted payoffs [5, 21, 20, 23]. Their applications range from logic, verification and finite automata theory [6, 12] to decision-making [22, 24] and algorithm design [3].

Along with this specialized research, in [9, 10] Gimbert and Zielonka undertook a thorough study of positionally determined payoffs in general. In [9] they showed that all the so-called *fairly mixing* payoffs are positionally determined. They also demonstrated that virtually all classical positionally determined payoffs are fairly mixing. Next, in [10] they established a property of payoffs which is *equivalent* to positional determinacy. Despite being rather technical, this property has a remarkable feature: if a payoff does not satisfy it, then this payoff violates positional determinacy in some *one-player* game graph (where one of the players owns all the nodes). As Gimbert and Zielonka indicate, this means that to establish positional determinacy of a payoff it is enough to do so only for one-player game graphs.

One could try to gain more understanding about positionally determined payoffs that satisfy certain additional requirements. Of course, this is interesting only if there are practically important positionally determined payoffs that satisfy these requirements. One such requirement studied in the literature is called *prefix-independence* [4, 8]. A payoff is prefix-independent if it is invariant under throwing away any finite prefix from an infinite sequence of edge labels. For instance, the parity and the mean payoffs are prefix-independent.

In [9] Gimbert and Zielonka briefly mention another interesting additional requirement, namely, *continuity.* They observe that the multi-discounted payoffs are continuous (they utilize this in showing that the multi-discounted payoffs are fairly mixing). In this paper we study continuous positionally determined payoffs in more detail. Continuity of a payoff,

[1] In particular, a node in which the game has started.

loosely speaking, means that its range converges to just a single point as more and more initial characters of an infinite sequence of edge labels are getting fixed. This contrasts with prefix-independent payoffs (such as the parity and the mean payoffs), for which any initial finite segment is irrelevant. Thus, continuity serves as a natural property which separates the multi-discounted payoffs from the other classical positionally determined payoffs. This is our main motivation to study continuous positionally determined payoffs in general, besides the general importance of the notion of continuity.

We show that for continuous payoff positional determinacy is equivalent to a simple property which we call *prefix-monotonicity*. Loosely speaking, prefix-monotonicity means the result of a comparison of the payoff on two infinite sequences of labels does not change after appending or deleting the same finite prefix. In fact, we prove this result in three different ways, using three major techniques of establishing positional determinacy:

- *An inductive argument.* Here we use a sufficient condition of Gimbert and Zielonka [9], which is proved by induction on the number of edges of a game graph. This type of argument goes back to a paper of Ehrenfeucht and Mycielski [5], where they provide an inductive proof of the positional determinacy of the Mean Payoff Games.
- *A fixed point argument.* Then we give a proof which uses a fixed point approach due to Shapley [23]. Shapley's technique is a standard way of establishing positional determinacy of Discounted Games. In this argument one derives positional determinacy from the existence of a solution to a certain system of equations (sometimes called *Bellman's equations*). In turn, to establish the existence of a solution one uses Banach's fixed point theorem.
- *A strategy improvement argument.* For Discounted Games the existence of a solution to Bellman's equations can also be proved by *strategy improvement.* This technique goes back to Howard [16]; for its thorough treatment (as well as for its applications to other payoffs) we refer the reader to [7]. We generalize it to arbitrary continuous positionally determined payoffs.

The simplest way to obtain our main result is via the inductive argument (at the cost of appealing without a proof to the results of Gimbert and Zielonka). We provide two other proofs for the following reasons.

First, they have applications (and it is unclear how to get these applications within the framework of the inductive approach). The fixed point approach provides a precise understanding of what do continuous positionally determined payoffs look like in general. In the full version of this paper [19] we use this to answer a question of Gimbert [8] regarding positional determinacy in more general *stochastic* games. In turn, the strategy improvement approach has algorithmic consequences. More specifically, we show that a problem of finding a pair of optimal positional strategies is solvable in randomized subexponential time for any continuous positionally determined payoff.

Second, as far as we know, these two approaches were never used in such an abstract setting before. Thus, we believe that our paper makes a useful addition to these approaches from a technical viewpoint. For example, the main problem for the fixed point approach is to identify a metric with which one can carry out the same "contracting argument" as in the case of multi-discounted payoffs. To solve it, we obtain a result of independent interest about compositions of continuous functions. As for the strategy improvement approach, our main contribution is a generalization of such well-established tools as "modified costs" and "potential transformation lemma" [15, Lemma 3.6].

Organization of the paper. In Section 2 we formalize the concepts discussed in the introduction. Then in Sections 3–6 we expose our results in more detail. In Section 7 we indicate some possible future directions. Most of the proofs are omitted due to space constraints. In this version we provide only one of the three proofs of our main result completely (namely, one by the induction argument). Missing proofs can be found in the full version of this paper [19].

2 Preliminaries

We denote the function composition by $\circ$.

Sets and sequences. For two sets A and B by A^B we denote the set of all functions from B to A (sometime we will interpret A^B as the set of vectors consisting of elements of A and with coordinates indexed by elements of B). We write $C = A \sqcup B$ for three sets A, B, C if A and B are disjoint and $C = A \cup B$.

For a set A by A^* we denote the set of all finite sequences of elements of A and by A^ω we denote the set of all infinite sequences of elements of A. For $w \in A^*$ we let $|w|$ be the length of w. For $\alpha \in A^\omega$ we let $|\alpha| = \infty$.

For $u \in A^*$ and $v \in A^* \cup A^\omega$ we let uv denote the concatenation of u and v. We call $u \in A^*$ a prefix of $v \in A^* \cup A^\omega$ if for some $w \in A^* \cup A^\omega$ we have $v = uw$. For $w \in A^*$ by wA^ω we denote the set $\{w\alpha \mid \alpha \in A^\omega\}$. Alternatively, wA^ω is the set of all $\beta \in A^\omega$ such that w is a prefix of β.

For $u \in A^*$ and $k \in \mathbb{N}$ we use a notation

$$u^k = \underbrace{uu\ldots u}_{k \text{ times}}.$$

In turn, we let $u^\omega \in A^\omega$ be a unique element of A^ω such that u^k is a prefix of u^ω for every $k \in \mathbb{N}$. We call $\alpha \in A^\omega$ ultimately periodic if α is a concatenation of u and v^ω for some $u, v \in A^*$.

Graphs notation. By a finite directed graph G we mean a pair $G = (V, E)$ of two finite sets V and E equipped with two functions $\mathsf{source}, \mathsf{target} \colon E \to V$. Elements of V are called nodes of G and elements of E are called edges of G. For an edge $e \in E$ we understand $\mathsf{source}(e)$ (respectively, $\mathsf{target}(e)$) as the node in which e starts (respectively, ends). We allow parallel edges; i.e., there might be two distinct edges $e, e' \in E$ with $\mathsf{source}(e) = \mathsf{source}(e')$, $\mathsf{target}(e) = \mathsf{target}(e')$. We allow self-loops as well (i.e., edges with $\mathsf{source}(e) = \mathsf{target}(e)$).

The out-degree of a node $a \in V$ is $|\{e \in E \mid \mathsf{source}(e) = a\}|$. A node $a \in V$ is called a sink if its out-degree is 0. We call a graph G sinkless if there are no sinks in G.

A path in G is a non-empty (finite or infinite) sequence of edges of G with a property that $\mathsf{target}(e) = \mathsf{source}(e')$ for any two consecutive edges e and e' from the sequence. For a path p we define $\mathsf{source}(p) = \mathsf{source}(e)$, where e is the first edge of p. For a finite path p we define $\mathsf{target}(p) = \mathsf{target}(e')$, where e' is the last edge of p.

For technical convenience we also consider 0-length paths. Each 0-length path is associated with some node of G (so that there are $|V|$ different 0-length paths). For a 0-length path p, associated with $a \in V$, we define $\mathsf{source}(p) = \mathsf{target}(p) = a$.

When we write pq for two paths p and q we mean the concatenation of p and q (viewed as sequences of edges). Of course, this is well-defined only if p is finite. Note that pq is not necessarily a path. Namely, pq is a path if and only if $\mathsf{target}(p) = \mathsf{source}(q)$.

2.1 Deterministic infinite duration games on finite directed graphs

Mechanics of the game. By a *game graph* we mean a sinkless finite directed graph $G = \langle V, E, \mathsf{source}, \mathsf{target}\rangle$, equipped with two sets V_{Max} and V_{Min} such that $V = V_{\mathrm{Max}} \sqcup V_{\mathrm{Min}}$.

A game graph $G = \langle V = V_{\mathrm{Max}} \sqcup V_{\mathrm{Min}}, E, \mathsf{source}, \mathsf{target}\rangle$ induces a so-called *infinite duration game* (IDG for short) on G. The game is always between two players called Max and Min. Positions of the game are finite paths in G (informally, these are possible finite trajectories of the token). We call a finite path p a Max's (a Min's) position if $\mathsf{target}(p) \in V_{\mathrm{Max}}$ (if $\mathsf{target}(p) \in V_{\mathrm{Min}}$). Max makes moves in Max's positions and Min makes moves in Min's positions. We do not indicate any position as the starting one – it can be any node of G.

The set of moves available at a position p is the set $\{e \in E \mid \mathsf{source}(e) = \mathsf{target}(p)\}$. A move e from a position p leads to a position pe.

A Max's strategy σ in a game graph G is a mapping assigning to every Max's position p a move available at p. Similarly, a Min's strategy τ in a game graph G is a mapping assigning to every Min's position p a move available at p.

Let $\mathcal{P} = e_1e_2e_3\dots$ be an infinite path in G. We say that $\mathcal{P}$ is *consistent* with a Max's strategy σ if the following conditions hold:

- if $s = \mathsf{source}(\mathcal{P}) \in V_{\mathrm{Max}}$, then $\sigma(s) = e_1$;
- for every $i \geq 1$ it holds that $\mathsf{target}(e_1e_2\dots e_i) \in V_{\mathrm{Max}} \implies e_{i+1} = \sigma(e_1e_2\dots e_i)$.

For $a \in V$ and for a Max's strategy σ we let $\mathsf{Cons}(a, \sigma)$ be a set of all infinite paths in G that start in a and are consistent with σ. We use similar terminology and notation for strategies of Min.

Given a Max's strategy σ, a Min's strategy τ and $a \in V$, we let *the play of σ and τ from a* be a unique element of the intersection $\mathsf{Cons}(a, \sigma) \cap \mathsf{Cons}(a, \tau)$. The play of σ and τ from a is denoted by $\mathcal{P}_a^{\sigma,\tau}$.

Positional strategies. A Max's strategy σ in a game graph $G = \langle V = V_{\mathrm{Max}} \sqcup V_{\mathrm{Min}}, E, \mathsf{source}, \mathsf{target}\rangle$ is called *positional* if $\sigma(p) = \sigma(q)$ for all finite paths p and q in G with $\mathsf{target}(p) = \mathsf{target}(q) \in V_{\mathrm{Max}}$. Clearly, a Max's positional strategy σ can be represented as a mapping $\sigma\colon V_{\mathrm{Max}} \to E$ satisfying $\mathsf{source}(\sigma(u)) = u$ for all $u \in V_{\mathrm{Max}}$. We define Min's positional strategies analogously.

We call an edge $e \in E$ *consistent* with a Max's positional strategy σ if either $\mathsf{source}(e) \in V_{\mathrm{Min}}$ or $\mathsf{source}(e) \in V_{\mathrm{Max}}, e = \sigma(\mathsf{source}(e))$. We denote the set of edges that are consistent with σ by E^σ. If τ is a Min's positional strategy, then we say that an edge $e \in E$ is consistent with τ if either $\mathsf{source}(e) \in V_{\mathrm{Max}}$ or $\mathsf{source}(e) \in V_{\mathrm{Min}}, e = \tau(\mathsf{source}(e))$. The set of edges that are consistent with a Min's positional strategy τ is denoted by E_τ.

Labels and payoffs. Let A be a finite set. A game graph $G = \langle, V = V_{\mathrm{Max}} \sqcup V_{\mathrm{Min}}, E, \mathsf{source}, \mathsf{target}\rangle$ equipped with a function $\mathsf{lab}\colon E \to A$ is called an *A-labeled game graph.* If $p = e_1e_2e_3\dots$ is a (finite or infinite) path in an A-labeled game graph $G = \langle V = V_{\mathrm{Max}} \sqcup V_{\mathrm{Min}}, E, \mathsf{source}, \mathsf{target}, \mathsf{lab}\rangle$, we define $\mathsf{lab}(p) = \mathsf{lab}(e_1)\mathsf{lab}(e_2)\mathsf{lab}(e_3)\dots \in A^* \cup A^\omega$. A *payoff* is a bounded function from A^ω to $\mathbb{R}$. Some papers allow A to be infinite and consider only infinite sequences that contain finitely many elements of A (as any game graph contains only finitely many labels). So basically we just have to deal with finite subsets of A, and this can be done with our approach.

Values, optimal strategies and equilibria. Let A be a finite set, $\varphi\colon A^\omega \to \mathbb{R}$ be a payoff and $G = \langle V = V_{\text{Max}} \sqcup V_{\text{Min}}, E, \textsf{source}, \textsf{target}, \textsf{lab}\rangle$ be an A-labeled game graph. Take a Max's strategy σ in G. *The value* of σ in a node $a \in V$ is the following quantity:

$$\textsf{Val}[\sigma](a) = \inf \varphi \circ \textsf{lab}\big(\textsf{Cons}(a, \sigma)\big).$$

Similarly, if τ is a Min's strategy in G, then the value of τ in a node $a \in V$ is the following quantity:

$$\textsf{Val}[\tau](a) = \sup \varphi \circ \textsf{lab}\big(\textsf{Cons}(a, \tau)\big).$$

A Max's strategy σ is called *optimal* if $\textsf{Val}[\sigma](a) \geq \textsf{Val}[\sigma'](a)$ for any $a \in V$ and for any Max's strategy σ'. Similarly, A Min's strategy τ is called *optimal* if $\textsf{Val}[\tau](a) \leq \textsf{Val}[\tau'](a)$ for any $a \in V$ and for any Min's strategy τ'.

Observe that for any Max's strategy σ, for any Min's strategy τ and for any $a \in V$ we have:

$$\textsf{Val}[\sigma](a) \leq \varphi \circ \textsf{lab}\big(\mathcal{P}_a^{\sigma,\tau}\big) \leq \textsf{Val}[\tau](a).$$

In particular, this inequality gives us the following. If a pair (σ, τ) of a Max's strategy σ and a Min's strategy τ is such that $\textsf{Val}[\sigma](a) = \textsf{Val}[\tau](a)$ for every $a \in V$, then both σ and τ are optimal for their players. We call any pair (σ, τ) with $\textsf{Val}[\sigma](a) = \textsf{Val}[\tau](a)$ for every $a \in V$ an *equilibrium*[2]. In fact, if at least one equilibrium exists, then the following holds: the Cartesian product of the set of the optimal strategies of Max and the set of the optimal strategies of Min is exactly the set of equilibria. We say that φ is *determined* if in every A-labeled game graph there exists an equilibrium (with respect to φ).

Positionally determined payoffs . Let A be a finite set and $\varphi\colon A^\omega \to \mathbb{R}$ be a payoff. We call φ *positionally determined* if all A-labeled game graphs have (with respect to φ) an equilibrium consisting of two positional strategies.

▶ **Proposition 1.** *If A is a finite set, $\varphi\colon A^\omega \to \mathbb{R}$ is a positionally determined payoff and $g\colon \varphi(A^\omega) \to \mathbb{R}$ is a non-decreasing*[3] *function, then $g \circ \varphi$ is a positionally determined payoff.*

2.2 Continuous payoffs

For a finite set A, we consider the set A^ω as a topological space. Namely, we take the discrete topology on A and the corresponding product topology on A^ω. In this product topology open sets are sets of the form

$$\mathcal{S} = \bigcup_{u \in S} uA^\omega,$$

where $S \subseteq A^*$. When we say that a payoff $\varphi\colon A^\omega \to \mathbb{R}$ is *continuous* we always mean continuity with respect to this product topology (and with respect to the standard topology on $\mathbb{R}$). The following proposition gives a convenient way to establish continuity of payoffs.

[2] This definition is equivalent to a more standard one: (σ, τ) is an equilibrium if and only if σ is a "best response" to τ in every node, and vice versa.

[3] Throughout the paper we call a function $f\colon S \to \mathbb{R}, S \subseteq \mathbb{R}$ non-decreasing if for all $x, y \in S$ with $x \leq y$ we have $f(x) \leq f(y)$.

▶ **Proposition 2.** *Let A be a finite set. A payoff $\varphi\colon A^\omega \to \mathbb{R}$ is continuous if and only if for any $\alpha \in A^\omega$ and for any infinite sequence $\{\beta_n\}_{n=1}^\infty$ of elements of A^ω the following holds. If for all $n \geq 1$ the sequences α and β_n coincide in the first n elements, then $\lim\limits_{n\to\infty} \varphi(\beta_n)$ exists and equals $\varphi(\alpha)$.*

For a finite A by Tychonoff's theorem the space A^ω is compact (because any finite set A with the discrete topology is compact). This has the following consequence which is important for this paper: if $\varphi\colon A^\omega \to \mathbb{R}$ is a continuous payoff, then $\varphi(A^\omega)$ is a compact subset of $\mathbb{R}$.

3 Statement of the Main Result and Preliminary Discussion

Our main result establishes a simple property which is equivalent to positional determinacy for continuous payoffs.

▶ **Definition 3.** *Let A be a finite set. A payoff $\varphi\colon A^\omega \to \mathbb{R}$ is called* ***prefix-monotone*** *if there are no $u, v \in A^*$, $\beta, \gamma \in A^\omega$ such that $\varphi(u\beta) > \varphi(u\gamma)$ and $\varphi(v\beta) < \varphi(v\gamma)$.*

(One can note that prefix-independence trivially implies prefix-monotonicity. On the other hand, no prefix-independent payoff which takes at least 2 values is continuous.)

▶ **Theorem 4.** *Let A be a finite set and $\varphi\colon A^\omega \to \mathbb{R}$ be a continuous payoff. Then φ is positionally determined if and only if φ is prefix-monotone.*

The fact that any continuous positionally determined payoff must be prefix-monotone[4] is proved in Appendix A. Three different proofs of the "if" part of Theorem 4 are discussed in, respectively, Sections 4, 5 and 6. Before going into the proofs, let us discuss the notions of continuity and prefix-monotonicity by means of the multi-discounted payoffs.

▶ **Definition 5.** *A payoff $\varphi\colon A^\omega \to \mathbb{R}$ for a finite set A is* ***multi-discounted*** *if there are functions $\lambda\colon A \to [0,1)$ and $w\colon A \to \mathbb{R}$ such that*

$$\varphi(a_1 a_2 a_3 \ldots) = \sum_{n=1}^{\infty} \lambda(a_1) \cdot \ldots \cdot \lambda(a_{n-1}) \cdot w(a_n) \tag{1}$$

for all $a_1 a_2 a_3 \ldots \in A^\omega$.

A few technical remarks: since the set A is finite, the coefficients $\lambda(a)$ are bounded away from 1 uniformly over $a \in A$. This ensures that the series (1) converges. In fact, this means that a tail of this series converges to 0 uniformly over $a_1 a_2 a_3 \ldots \in A^\omega$. Thus, the multi-discounted payoffs are continuous. As the multi-discounted payoffs are positionally determined, by Theorem 4 they also must be prefix-monotone. Of course, prefix-monotonicity of the multi-discounted payoffs can be established without Theorem 4. Indeed, from (1) it is easy to derive that $\varphi(a\beta) - \varphi(a\gamma) = \lambda(a) \cdot (\varphi(\beta) - \varphi(\gamma))$ for all $a \in A, \beta, \gamma \in A^\omega$. Due to the condition $\lambda(a) \geq 0$, we have that $\varphi(a\beta) > \varphi(a\gamma)$ implies that $\varphi(\beta) > \varphi(\gamma)$. Moreover, the same holds if we append more than one character to β and γ. Hence it is impossible to simultaneously have $\varphi(u\beta) > \varphi(u\gamma)$ and $\varphi(v\beta) < \varphi(v\gamma)$ for $u, v \in A^*$, as required in the definition of prefix-monotonicity.

[4] Here it is crucial that in our definition of positional determinacy we require that some positional strategy is optimal for all the nodes. Allowing each starting node to have its own optimal positional strategy gives us a weaker, "non-uniform" version of positional determinacy. It is not clear whether non-uniform positional determinacy implies prefix-monotonicity. At the same time, we are not even aware of a payoff which is positional only "non-uniformly".

4 Inductive Argument

Here we show that any continuous prefix-monotone payoff is positionally determined using a sufficient condition of Gimbert and Zielonka [9, Theorem 1], which, in turn, is proved by an inductive argument. As Gimbert and Zielonka indicate [9, Lemma 2], their sufficient condition takes the following form for continuous payoffs[5].

▶ **Proposition 6.** *Let A be a finite set. Any continuous payoff $\varphi\colon A^\omega \to \mathbb{R}$, satisfying the following two conditions:*

- ***(a)** for all $u \in A^*$ and $\alpha, \beta \in A^\omega$ we have that $\varphi(\alpha) \le \varphi(\beta) \implies \varphi(u\alpha) \le \varphi(u\beta)$;*
- ***(b)** for all non-empty $u \in A^*$ and for all $\alpha \in A^\omega$ we have that*

$$\min\{\varphi(u^\omega), \varphi(\alpha)\} \le \varphi(u\alpha) \le \max\{\varphi(u^\omega), \varphi(\alpha)\};$$

is positionally determined.

We observe that one can get rid of the condition ***(b)*** in this Proposition.

▶ **Proposition 7.** *For continuous payoffs the condition **(a)** of Proposition 6 implies the condition **(b)** of Proposition 6.*

Proof. See Appendix B. ◀

So to establish positional determinacy of a continuous payoff it is enough to demonstrate that this payoff satisfies the condition ***(a)*** of Proposition 6. Let us now reformulate this condition using the following definition.

▶ **Definition 8.** *Let A be a finite set. A payoff $\varphi\colon A^\omega \to \mathbb{R}$ is called **shift-deterministic** if for all $a \in A, \beta, \gamma \in A^\omega$ we have $\varphi(\beta) = \varphi(\gamma) \implies \varphi(a\beta) = \varphi(a\gamma)$.*

▶ **Observation 9.** *Let A be a finite set. A payoff $\varphi\colon A^\omega \to \mathbb{R}$ satisfies the condition **(a)** of Proposition 6 if and only if φ is prefix-monotone and shift-deterministic.*

The above discussion gives the following sufficient condition for positional determinacy.

▶ **Proposition 10.** *Let A be a finite set. Any continuous prefix-monotone shift-deterministic payoff $\varphi\colon A^\omega \to \mathbb{R}$ is positionally determined.*

Still, some argument is needed for continuous prefix-monotone payoffs that are not shift-deterministic. To tie up loose ends we prove the following:

▶ **Proposition 11.** *Let A be a finite set and let $\varphi\colon A^\omega \to \mathbb{R}$ be a continuous prefix-monotone payoff. Then $\varphi = g \circ \psi$ for some continuous prefix-monotone shift-deterministic payoff $\psi\colon A^\omega \to \mathbb{R}$ and for some continuous[6] non-decreasing function $g\colon \psi(A^\omega) \to \mathbb{R}$.*

Proof. See Appendix C. ◀

Due to Proposition 1 this finishes our first proof of Theorem 4. In fact, we do not need continuity of g here, but it will be useful later.

[5] Lemma 2 can only be found in the HAL version of their paper.

[6] Throughout the paper we call a function $f\colon S \to \mathbb{R}$, $S \subseteq \mathbb{R}^n$ continuous if f is continuous with respect to a restriction of the standard topology of $\mathbb{R}^n$ to S.

5 Fixed point argument

Here we present a way of establishing positional determinacy of continuous prefix-monotone shift-deterministic payoffs (Proposition 10) via a fixed point argument. Together with Proposition 11 this constitutes our second proof of Theorem 4.

Obviously, for any shift-deterministic payoff $\varphi\colon A^\omega \to \mathbb{R}$ and for any $a \in A$ there is a unique function $\mathsf{shift}[a,\varphi]\colon \varphi(A^\omega) \to \varphi(A^\omega)$ such that $\mathsf{shift}[a,\varphi]\big(\varphi(\beta)\big) = \varphi(a\beta)$ for all $\beta \in A^\omega$.

▶ **Observation 12.** *A shift-deterministic payoff $\varphi\colon A^\omega \to \mathbb{R}$ is prefix-monotone if and only if $\mathsf{shift}[a,\varphi]$ is non-decreasing for every $a \in A$.*

We use this notation to introduce the so-called *Bellman's equations*, playing a key role in our fixed point argument.

▶ **Definition 13.** *Let A be a finite set, $\varphi\colon A^\omega \to \mathbb{R}$ be a shift-deterministic payoff and $G = \langle V = V_{\mathrm{Max}} \sqcup V_{\mathrm{Min}}, E, \mathsf{source}, \mathsf{target}, \mathsf{lab}\rangle$ be an A-labeled game graph.*

*The following equations in $\mathbf{x} \in \varphi(A^\omega)^V$ are called **Bellman's equations** for φ in G:*

$$\mathbf{x}_u = \max_{e \in E, \mathsf{source}(e) = u} \mathsf{shift}[\mathsf{lab}(e),\varphi]\big(\mathbf{x}_{\mathsf{target}(e)}\big), \qquad \textit{for } u \in V_{\mathrm{Max}}, \tag{2}$$

$$\mathbf{x}_u = \min_{e \in E, \mathsf{source}(e) = u} \mathsf{shift}[\mathsf{lab}(e),\varphi]\big(\mathbf{x}_{\mathsf{target}(e)}\big), \qquad \textit{for } u \in V_{\mathrm{Min}}. \tag{3}$$

The most important step of our argument is to show the existence of a solution to Bellman's equations.

▶ **Proposition 14.** *For any finite set A, for any continuous prefix-monotone shift-deterministic payoff $\varphi\colon A^\omega \to \mathbb{R}$ and for any A-labeled game graph G there exists a solution to Bellman's equations for φ in G.*

(One can also show the uniqueness of a solution, but we do not need this for the argument).

This proposition requires some additional work, and we first discuss how to derive positional determinacy of continuous prefix-monotone shift-deterministic payoffs from it. Assume that we are give a solution $\mathbf{x}$ to (2–3). How can one extract an equilibrium of positional strategies from it? For that we take any pair of positional strategies that use only $\mathbf{x}$-*tight* edges. Now, an edge e is $\mathbf{x}$-tight if $\mathbf{x}_{\mathsf{source}(e)} = \mathsf{shift}[a,\varphi](\mathbf{x}_{\mathsf{target}(e)})$. Note that each node must contain an out-going $\mathbf{x}$-tight edge (this will be any edge on which the maximum/minimum in (2–3) is attained for this node). So clearly each player has at least one positional strategy which only uses $\mathbf{x}$-tight edges. It remains to show that for continuous prefix-monotone shift-deterministic φ any two such strategies of the players form an equilibrium.

▶ **Lemma 15.** *If A is a finite set, $\varphi\colon A^\omega \to \mathbb{R}$ is a continuous prefix-monotone shift-deterministic payoff, and $\mathbf{x} \in \varphi(A^\omega)^V$ is a solution to (2–3) for an A-labeled game graph $G = \langle V = V_{\mathrm{Max}} \sqcup V_{\mathrm{Min}}, E, \mathsf{source}, \mathsf{target}, \mathsf{lab}\rangle$, then the following holds. Let σ^* be a positional strategy of Max and τ^* be a positional strategy of Min such that $\sigma^*(V_{\mathrm{Max}})$ and $\tau^*(V_{\mathrm{Min}})$ consist only of $\mathbf{x}$-tight edges. Then (σ^*, τ^*) is an equilibrium in G.*

We now proceed to details of our proof of Proposition 14. Consider a function $T\colon \varphi(A^\omega)^V \to \varphi(A^\omega)^V$, mapping $\mathbf{x} \in \varphi(A^\omega)^V$ to the vector of the right-hand sides of (2–3). We should argue that T has a fixed point. For that we will construct a continuous metric $D\colon \varphi(A^\omega)^V \times \varphi(A^\omega)^V \to [0,+\infty)$ with respect to which T is *contracting*. More precisely, $D(T\mathbf{x}, T\mathbf{y})$ will always be smaller than $D(\mathbf{x},\mathbf{y})$ as long as $\mathbf{x}$ and $\mathbf{y}$ are distinct. Due to the compactness of the domain of T this will prove that T has a fixed point.

Now, to construct such D we show that for continuous shift-deterministic φ there must be a continuous metric $d\colon \varphi(A^\omega) \times \varphi(A^\omega) \to [0, +\infty)$ such that for all $a \in A$ the function $\mathsf{shift}[a, \varphi]$ is d-contracting. Once we have such d, we let $D(\mathbf{x}, \mathbf{y})$ be the maximum of $d(\mathbf{x}_a, \mathbf{y}_a)$ over $a \in V$. Checking that T is contracting with respect to such D will be rather straightforward (technically, we will need an additional property of d which can be derived from the prefix-monotonicity of φ).

The main technical challenge is to prove the existence of d. In the full version of this paper we do so via the following general fact about compositions of continuous functions.

▶ **Theorem 16.** *Let $K \subseteq \mathbb{R}$ be a compact set, $m \geq 1$ be a natural number and $f_1, \ldots, f_m\colon K \to K$ be m continuous functions. Then the following two conditions are equivalent:*

- ***(a)** for any $a_1a_2a_3 \ldots \in \{1, 2, \ldots, m\}^\omega$ we have $\lim_{n\to\infty} \mathsf{diam}\big(f_{a_1} \circ f_{a_2} \circ \ldots \circ f_{a_n}(K)\big) = 0$ (by $\mathsf{diam}(S)$ for $S \subseteq \mathbb{R}$ we mean $\sup_{x,y\in S} |x - y|$);*
- ***(b)** there exists a continuous metric $d\colon K \times K \to [0, +\infty)$ such that $f_1, f_2, \ldots, f_m$ are all d-contracting (a function $h\colon K \to K$ is called d-contracting if for all $x, y \in K$ with $x \neq y$ we have $d(h(x), h(y)) < d(x, y)$).*

*If $f_1, \ldots, f_m$ are non-decreasing, then one can strengthen item **(b)** by demanding that d satisfies the following property: for all $x, y, s, t \in K$ with $x \leq s \leq t \leq y$ we have $d(s, t) \leq d(x, y)$.*

Namely, we apply this theorem to the functions $\mathsf{shift}[a, \varphi]$ for $a \in A$ (for that we first show that the continuity of φ implies that these functions satisfy item ***(a)*** of Theorem 16).

Applications of the fixed point technique

Theorem 16 additionally provides an exhaustive method of generating continuous positionally determined payoffs.

▶ **Theorem 17.** *Let m be a natural number. The set of continuous positionally determined payoffs from*[7] *$\{1, 2, \ldots, m\}^\omega$ to $\mathbb{R}$ coincides with the set of φ that can be obtained in the following 5 steps.*

- ***Step 1.** Take a compact set $K \subseteq \mathbb{R}$.*
- ***Step 2.** Take a continuous metric $d\colon K \times K \to [0, +\infty)$.*
- ***Step 3.** Take m non-decreasing d-contracting functions $f_1, f_2, \ldots, f_m\colon K \to K$ (they will automatically be continuous due to continuity of d).*
- ***Step 4.** Define $\psi\colon \{1, \ldots, m\}^\omega \to K$ so that*

$$\{\psi(a_1a_2a_3\ldots)\} = \bigcap_{n=1}^{\infty} f_{a_1} \circ f_{a_2} \circ \ldots \circ f_{a_n}(K)$$

  for every[8] *$a_1a_2a_3 \ldots \in \{1, 2, \ldots, m\}^\omega$.*
- ***Step 5.** Choose a continuous non-decreasing function $g\colon \psi(\{1, 2, \ldots, m\}^\omega) \to \mathbb{R}$ and set $\varphi = g \circ \psi$.*

▶ Remark 18. Recall that we did not use continuity of g from Proposition 11 in the inductive argument. It becomes important for Theorem 17 – otherwise we could not argue that all continuous positionally payoffs can be obtained in these 5 steps.

[7] Of course, in this theorem a set of labels can be any finite set, we let it be $\{1, 2, \ldots, m\}$ for some $m \in \mathbb{N}$ just to simplify the notation.

[8] Note that this intersection always consists of a single point due to Cantor's intersection theorem and item ***(a)*** of Theorem 16. This will also be $\lim_{n\to\infty} f_{a_1} \circ f_{a_2} \circ \ldots \circ f_{a_n}(x)$ for any $x \in K$.

We get the multi-discounted payoffs when the functions $f_1, f_2, \ldots, f_m$ are affine, each with the slope from $[0, 1)$. In this case they will be contracting with respect to a standard metric $d(x, y) = |x - y|$. We get the whole set of continuous positionally determined payoffs by relaxing the multi-discounted payoffs in the following three regards: ***(a)*** functions $f_1, f_2, \ldots, f_m$ do not have to be affine; ***(b)*** d can be an arbitrary continuous metric; ***(c)*** any continuous non-decreasing function g can be applied to a payoff.

We use Theorem 17 to construct a continuous positionally determined payoff which does not "reduce" to the multi-discounted ones, in a sense of the following definition.

▶ **Definition 19.** *Let A be a finite set, $\varphi, \psi \colon A^\omega \to \mathbb{R}$ be two payoffs, and G be an A-labeled game graph. We say that φ* ***positionally reduces*** *to ψ* ***inside*** *G if any pair of positional strategies in G which is an equilibrium for ψ is also an equilibrium for φ.*

This definition has an algorithmic motivation. Namely, note that finding a positional equilibrium for ψ in G is at least as hard as for φ, provided that φ reduces to ψ inside G. There are classical reductions from Parity to Mean Payoff games [17] and from Mean Payoff to Discounted games [25] that work in exactly this way. See also [11] for a reduction from *Priority* Mean Payoff games to Multi-Discounted games. As far as we know, our next proposition provides the first example of a positionally determined payoff which does not reduce to the multi-discounted ones in this sense.

▶ **Proposition 20.** *There exist a finite set A, a continuous positionally determined payoff $\varphi \colon A^\omega \to \mathbb{R}$ and an A-labeled game graph G such that there exists no multi-discounted payoff to which φ reduces inside G.*

Proposition 20 means, in particular, that there exists a continuous positionally determined payoff which differs from all the multi-discounted ones (as was stated in Section 3). This fact alone can be used to disprove a conjecture of Gimbert [8]. Namely, Gimbert conjectured the following: "Any payoff function which is positional for the class of non-stochastic one-player games is positional for the class of Markov decision processes". To show that this is not the case, in the full version of this paper [19] we establish that all continuous payoffs that are positionally determined in Markov decision processes are multi-discounted.

6 Strategy improvement argument

Here we establish the existence of a solution to Bellman's equations (Proposition 14) via the *strategy improvement*. This will yield our third proof of Theorem 4. We start with an observation that a vector of values of a positional strategy always gives a solution[9] to a restriction of Bellman's equations to edges that are consistent with this strategy.

▶ **Lemma 21.** *Let A be a finite set, $\varphi \colon A^\omega \to \mathbb{R}$ be a continuous prefix-monotone shift-deterministic payoff and $G = \langle V = V_{\mathrm{Max}} \sqcup V_{\mathrm{Min}}, E, \mathsf{source}, \mathsf{target}, \mathsf{lab}\rangle$ be an A-labeled game graph. Then for every positional strategy σ of Max in G we have:*

$$\mathsf{Val}[\sigma](u) = \mathsf{shift}[\mathsf{lab}(\sigma(u)), \varphi]\Big(\mathsf{Val}[\sigma](\mathsf{target}(\sigma(u)))\Big) \text{ for } u \in V_{\mathrm{Max}},$$

$$\mathsf{Val}[\sigma](u) = \min_{e \in E, \mathsf{source}(e) = u} \mathsf{shift}[\mathsf{lab}(e), \varphi]\Big(\mathsf{Val}[\sigma](\mathsf{target}(e))\Big) \text{ for } u \in V_{\mathrm{Min}}.$$

[9] Bellman's equations involve the functions $\mathsf{shift}[a, \varphi]$ for $a \in A$, and these functions are defined on $\varphi(A^\omega)$. So formally we should argue that the values of any strategy belong to $\varphi(A^\omega)$. Indeed, for continuous φ the set $\varphi(A^\omega)$ is compact and hence is closed, and all values are the infimums/supremums of some subsets of $\varphi(A^\omega)$.

Next, take a positional strategy σ of Max. If the vector $\{\mathsf{Val}[\sigma](u)\}_{u\in V}$ happens to be a solution to the Bellman's equations, then we are done. Otherwise by Lemma 21 there must exist an edge $e \in E$ with $\mathsf{source}(e) \in V_{\mathrm{Max}}$ such that $\mathsf{Val}[\sigma](\mathsf{source}(e)) < \mathsf{shift}[\mathsf{lab}(e), \varphi]\big(\mathsf{Val}[\sigma](\mathsf{target}(e))\big)$. We call edges satisfying this property *σ-violating*. We show that *switching* σ to any σ-violating edge gives us a positional strategy which *improves* σ.

▶ **Lemma 22.** *Let A be a finite set, $\varphi\colon A^\omega \to \mathbb{R}$ be a continuous prefix-monotone shift-deterministic payoff and $G = \langle V = V_{\mathrm{Max}} \sqcup V_{\mathrm{Min}}, E, \mathsf{source}, \mathsf{target}, \mathsf{lab}\rangle$ be an A-labeled game graph. Next, let σ be a positional strategy of Max in G. Assume that the vector $\mathsf{Val}[\sigma] = \{\mathsf{Val}[\sigma](u)\}_{u\in V}$ does not satisfy (2–3) and let $e' \in E$ be any σ-violating edge. Define a positional strategy σ' of Max as follows:*

$$\sigma'(u) = \begin{cases} e' & u = \mathsf{source}(e'), \\ \sigma(u) & \textit{otherwise.} \end{cases}$$

Then $\sum\limits_{u\in V} \mathsf{Val}[\sigma'](u) > \sum\limits_{u\in V} \mathsf{Val}[\sigma](u)$.

By this lemma, a Max's positional strategy σ^* maximizing the quantity $\sum_{u\in V} \mathsf{Val}[\sigma](u)$ (over positional strategies σ of Max) gives a solution to (2–3). Such σ^* exists just because there are only finitely many positional strategies of Max. This finishes our strategy improvement proof of Proposition 14. Let us note that the same argument can be carried out with positional strategies of Min (via analogues of Lemma 21 and Lemma 22 for Min).

Applications of the strategy improvement technique

In this subsection we discuss implications of our strategy improvement argument to the *strategy synthesis problem.* Strategy synthesis for a positionally determined payoff φ is an algorithmic problem of finding an equilibrium (with respect to φ) of two positional strategies for a given game graph. It is classical that strategy synthesis for classical positionally determined payoffs admits a randomized algorithm which is subexponential in the number of nodes [14, 1]. We obtain the same subexponential bound for all continuous positionally determined payoffs. From a technical viewpoint, we just observe that a technique which was used for classical positionally determined payoffs is applicable in a more general setting. Specifically, we use a framework of recursively local-global functions due to Björklund and Vorobyov [1].

Let us start with an observation that for continuous positionally determined shift-deterministic payoffs a non-optimal positional strategy can always be improved by changing it just in a single node.

▶ **Proposition 23.** *Let A be a finite set and $\varphi\colon A^\omega \to \mathbb{R}$ be a continuous positionally determined shift-deterministic payoff. Then for any A-labeled game graph $G = \langle V = V_{\mathrm{Max}} \sqcup V_{\mathrm{Min}}, E, \mathsf{source}, \mathsf{target}, \mathsf{lab}\rangle$ the following two conditions hold:*

- *if σ is a non-optimal positional strategy of Max in G, then in G there exists a Max's positional strategy σ' such that $|\{u \in V_{\mathrm{Max}} \mid \sigma(u) \neq \sigma'(u)\}| = 1$ and $\sum_{u\in V} \mathsf{Val}[\sigma'](u) > \sum_{u\in V} \mathsf{Val}[\sigma](u)$;*
- *if τ is a non-optimal positional strategy of Min in G, then in G there exists a Min's positional strategy τ' such that $|\{u \in V_{\mathrm{Min}} \mid \tau(u) \neq \tau'(u)\}| = 1$ and $\sum_{u\in V} \mathsf{Val}[\tau'](u) < \sum_{u\in V} \mathsf{Val}[\tau](u)$.*

It is instructive to visualize this proposition by imagining the set of positional strategies of one of the players (say, Max) as a *hypercube.* Namely, in this hypercube there will be as many dimensions as there are nodes of Max. A coordinate corresponding to a node $u \in V_{\mathrm{Max}}$

will take values in the set of edges that start at u. Obviously, vertices of such hypercube are in a one-to-one correspondence with positional strategies of Max. Let us call two vertices *neighbors* of each other if they differ in exactly one coordinate. Now, Proposition 23 means in this language the following: any vertex σ, maximizing $\sum_{u\in V} \mathsf{Val}[\sigma](u)$ over its neighbors, also maximizes this quantity over the *whole* hypercube.

So an optimization problem of maximizing $\sum_{u\in V} \mathsf{Val}[\sigma](u)$ (equivalently, finding an optimal positional strategy of Max) has the following remarkable feature: all its *local* maxima are also *global*. For positional strategies of Min the same holds for the minima. Optimization problems with this feature are in a focus of numerous works, starting from a classical area of convex optimization.

Observe that in our case this local-global property is *recursive*; i.e., it holds for any restriction to a *subcube* of our hypercube. Indeed, subcubes correspond to subgraphs of our initial game graph, and for any subgraph we still have Proposition 23. Björklund and Vorobyov [1] noticed that a similar phenomenon occurs for all classical positionally determined payoffs. In turn, they showed that any optimization problem on a hypercube with this recursive local-global property admits a randomized algorithm which is subexponential in the dimension of a hypercube. In our case this yields a randomized algorithm for the strategy synthesis problem which is subexponential in the number of nodes of a game graph.

Still, this only applies to continuous payoffs that are shift-deterministic (as we have Proposition 23 only for shift-deterministic payoffs). One more issue is that we did not specify how our payoffs are represented. We overcome these difficulties in the following result.

▶ **Theorem 24.** *Let A be a finite set and $\varphi\colon A^\omega \to \mathbb{R}$ be a continuous positionally determined payoff. Consider an oracle which for given $u, v, a, b \in A^*$ tells, whether there exists $w \in A^*$ such that $\varphi(wu(v)^\omega) > \varphi(wa(b)^\omega)$. There exists a randomized algorithm which with this oracle solves the strategy synthesis problem for φ in expected $e^{O\left(\log m+\sqrt{n\log m}\right)}$ time for game graphs with n nodes and m edges. In particular, every call to the oracle in the algorithm is for $u, v, a, b \in A^*$ that are of length $O(n)$, and the expected number of the calls is $e^{O\left(\log m+\sqrt{n\log m}\right)}$.*

So to deal with the issue of representation we assume a suitable oracle access to φ. Still, the oracle from Theorem 24 might look unmotivated. Here it is instructive to recall that all continuous positionally determined φ must be prefix-monotone. For prefix-monotone φ a formula $\exists w \in A^*\ \varphi(w\alpha) > \varphi(w\beta)$ defines a total preorder on A^ω, and our oracle just compares ultimately periodic sequences according to this preorder. In fact, it is easy to see that the formula $\exists w \in A^*\ \varphi(w\alpha) > \varphi(w\beta)$ defines a total preorder on A^ω if and *only if* φ is prefix-monotone. This indicates a fundamental role of this preorder for prefix-monotone φ and justifies a use of the corresponding oracle in Theorem 24. Let us note that $\left[\exists w \in A^*\ \varphi(w\alpha) > \varphi(w\beta)\right] \iff \varphi(\alpha) > \varphi(\beta)$ if φ is additionally shift-deterministic.

7 Discussion

As Gimbert and Zielonka show by their characterization of the class of positionally determined payoffs [10], positional determinacy can always be proved by an inductive argument. Does the same hold for two other techniques that we have considered in the paper – the fixed point technique and the strategy improvement technique? The answer is positive in the continuous case, so this suggests that the answer might also be positive at least in some other special cases, for instance, for prefix-independent payoffs. E.g., for the mean payoff, a major example of a prefix-independent positionally determined payoff, both the strategy improvement and the fixed point arguments are applicable [13, 18].

These questions are specifically interesting for the strategy improvement argument. Indeed, strategy improvement usually leads to subexponential-time (randomized) algorithms for the strategy synthesis. So this resonates with a question of how hard strategy synthesis for a positionally determined payoff can be. Loosely speaking, do we have this subexponential bound for all positionally determined payoffs (as we do, by Theorem 24, for all such payoffs that are additionally continuous)?

Finally, is it possible to characterize positionally determined payoffs more explicitly (say, as in Theorem 17)? This question sounds more approachable in special cases, and a natural special case to start is again the prefix-independent case.

References

1 Henrik Björklund and Sergei Vorobyov. Combinatorial structure and randomized subexponential algorithms for infinite games. *Theoretical Computer Science*, 349(3):347–360, 2005.

2 Patricia Bouyer, Stéphane Le Roux, Youssouf Oualhadj, Mickael Randour, and Pierre Vandenhove. Games where you can play optimally with arena-independent finite memory. In *31st International Conference on Concurrency Theory (CONCUR 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

3 Cristian S Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 252–263, 2017.

4 Thomas Colcombet and Damian Niwiński. On the positional determinacy of edge-labeled games. *Theoretical Computer Science*, 352(1-3):190–196, 2006.

5 Andrzej Ehrenfeucht and Jan Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8(2):109–113, 1979.

6 E Allen Emerson and Charanjit S Jutla. Tree automata, mu-calculus and determinacy. In *FoCS*, volume 91, pages 368–377. Citeseer, 1991.

7 John Fearnley. *Strategy iteration algorithms for games and Markov decision processes.* PhD thesis, University of Warwick, 2010.

8 Hugo Gimbert. Pure stationary optimal strategies in markov decision processes. In *Annual Symposium on Theoretical Aspects of Computer Science*, pages 200–211. Springer, 2007.

9 Hugo Gimbert and Wieslaw Zielonka. When can you play positionnaly? In I. Fiala, V. Koubek, and J. Kratochvil, editors, *Mathematical Foundations of Computer Science 2004*, Lecture Notes in Comp. Sci. 3153, pages 686–697, Prague, Czech Republic, 2004. Springer. URL: https://hal.archives-ouvertes.fr/hal-00160436.

10 Hugo Gimbert and Wiesław Zielonka. Games where you can play optimally without any memory. In *International Conference on Concurrency Theory*, pages 428–442. Springer, 2005.

11 Hugo Gimbert and Wieslaw Zielonka. Applying blackwell optimality: priority mean-payoff games as limits of multi-discounted games. In *Logic and automata*, pages 331–356, 2008.

12 Erich Gradel and Wolfgang Thomas. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Science & Business Media, 2002.

13 Vladimir A Gurvich, Alexander V Karzanov, and LG Khachivan. Cyclic games and an algorithm to find minimax cycle means in directed graphs. *USSR Computational Mathematics and Mathematical Physics*, 28(5):85–91, 1988.

14 Nir Halman. Simple stochastic games, parity games, mean payoff games and discounted payoff games are all lp-type problems. *Algorithmica*, 49(1):37–50, 2007.

15 Thomas Dueholm Hansen, Peter Bro Miltersen, and Uri Zwick. Strategy iteration is strongly polynomial for 2-player turn-based stochastic games with a constant discount factor. *Journal of the ACM (JACM)*, 60(1):1–16, 2013.

16 Ronald A Howard. *Dynamic programming and markov processes.* The M.I.T. Press, 1960.

17 Marcin Jurdziński. Deciding the winner in parity games is in up∩co-up. *Information Processing Letters*, 68(3):119–124, 1998.

18 Elon Kohlberg. Invariant half-lines of nonexpansive piecewise-linear transformations. *Mathematics of Operations Research*, 5(3):366–372, 1980.

19 Alexander Kozachinskiy. Continuous positional payoffs. *arXiv preprint*, 2020. arXiv:2012.09047.

20 Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

21 Andrzej W Mostowski. Games with forbidden positions. Technical Report 78, Uniwersytet Gdánski, Instytut Matematyki, 1991.

22 Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, 2014.

23 Lloyd S Shapley. Stochastic games. *Proceedings of the national academy of sciences*, 39(10):1095–1100, 1953.

24 Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

25 Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1-2):343–359, 1996.

A The "Only If" Part of Theorem 4

Assume that φ is not prefix-monotone. Then for some $u, v \in A^*$ and $\alpha, \beta \in A^\omega$ we have

$$\varphi(u\alpha) > \varphi(u\beta) \text{ and } \varphi(v\alpha) < \varphi(v\beta). \tag{4}$$

First, notice that by continuity of φ we may assume that α and β are ultimately periodic. Indeed, consider any two sequences $\{\alpha_n\}_{n\in\mathbb{N}}$ and $\{\beta_n\}_{n\in\mathbb{N}}$ of ultimately periodic sequences from A^ω such that α_n and α (respectively, β_n and β) have the same prefix of length n. Then from continuity of φ (by Proposition 2) we have:

$$\lim_{n\to\infty} \varphi(u\alpha_n) = \varphi(u\alpha), \qquad \lim_{n\to\infty} \varphi(v\alpha_n) = \varphi(v\alpha),$$
$$\lim_{n\to\infty} \varphi(u\beta_n) = \varphi(u\beta), \qquad \lim_{n\to\infty} \varphi(v\beta_n) = \varphi(v\beta).$$

So if u, v, α, β violate prefix-monotonicity, then so do u, v, α_n, β_n for some $n \in \mathbb{N}$.

Now, if α, β are ultimately periodic, then $\alpha = p(q)^\omega$ and $\beta = w(r)^\omega$ for some $p, q, w, r \in A^*$. Consider an A-labeled game graph from Figure 1 (all nodes there are owned by Max).

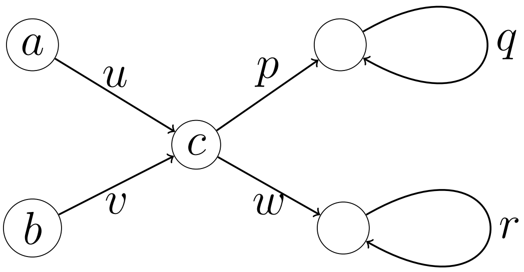


Figure 1 A game graph where φ is not positionally determined.

In this game graph there are two positional strategies of Max, one which from c goes by p and the other which goes from c by w. The first one is not optimal when the game starts in b, and the second one is not optimal when the game starts in a (because of (4)). So φ is not positionally determined in this game graph.

B Proof of Proposition 7

We only show that $\varphi(u\alpha) \le \max\{\varphi(u^\omega), \varphi(\alpha)\}$, the other inequality can be proved similarly. If $\varphi(u\alpha) \le \varphi(\alpha)$, then we are done. Assume now that $\varphi(u\alpha) > \varphi(\alpha)$. By repeatedly applying ***(a)*** we obtain $\varphi(u^{i+1}\alpha) \ge \varphi(u^i\alpha)$ for every $i \in \mathbb{N}$. In particular, for every $i \ge 1$ we get that $\varphi(u^i\alpha) \ge \varphi(u\alpha)$. By continuity of φ the limit of $\varphi(u^i\alpha)$ as $i \to \infty$ exists and equals $\varphi(u^\omega)$. Hence $\varphi(u^\omega) \ge \varphi(u\alpha)$.

C Proof of Proposition 11

Define a payoff $\psi\colon A^\omega \to \mathbb{R}$ as follows:

$$\psi(\gamma) = \sum_{w \in A^*} \left(\frac{1}{|A|+1}\right)^{|w|} \varphi(w\gamma), \qquad \gamma \in A^\omega. \tag{5}$$

First, why is ψ well-defined, i.e., why does this series converge? Since A^ω is compact, so is $\varphi(A^\omega) \subseteq \mathbb{R}$, because φ is continuous. Hence $\varphi(A^\omega) \subseteq [-W, W]$ for some $W > 0$ and (5) is bounded by the following absolutely converging series:

$$\sum_{w \in A^*} W \cdot \left(\frac{1}{|A|+1}\right)^{|w|}.$$

We shall show that ψ is continuous, prefix-monotone and shift-deterministic, and that $\varphi = g \circ \psi$ for some continuous non-decreasing $g\colon \psi(A^\omega) \to \mathbb{R}$.

Why is ψ continuous? Consider any $\alpha \in A^\omega$ and any infinite sequence $\{\beta_n\}_{n\in\mathbb{N}}$ of elements of A^ω such that for all n the sequences α and β_n coincide in the first n elements. We have to show that $\psi(\beta_n)$ converges to $\psi(\alpha)$ as $n \to \infty$. By definition:

$$\psi(\beta_n) = \sum_{w \in A^*} \left(\frac{1}{|A|+1}\right)^{|w|} \varphi(w\beta_n), \qquad \psi(\alpha) = \sum_{w \in A^*} \left(\frac{1}{|A|+1}\right)^{|w|} \varphi(w\alpha).$$

The first series, as we have seen, is bounded uniformly (in n) by an absolutely converging series. So it remains to note that the first series converges to the second one term-wise, by continuity of φ.

Why is ψ prefix-monotone? Let $\alpha, \beta \in A^\omega$. We have to show that either $\psi(u\alpha) \ge \psi(u\beta)$ for all $u \in A^*$ or $\psi(u\alpha) \le \psi(u\beta)$ for all $u \in A^*$.

Since φ is prefix-monotone, then either $\varphi(w\alpha) \ge \varphi(w\beta)$ for all $w \in A^*$ or $\varphi(w\alpha) \le \varphi(w\beta)$ for all $w \in A^*$. Up to swapping α and β we may assume that $\varphi(w\alpha) \ge \varphi(w\beta)$ for all $w \in A^*$. Then for any $u \in A^*$ the difference

$$\psi(u\alpha) - \psi(u\beta) = \sum_{w \in A^*} \left(\frac{1}{|A|+1}\right)^{|w|} \left[\varphi(wu\alpha) - \varphi(wu\beta)\right]$$

consists of non-negative terms. Hence $\psi(u\alpha) \ge \psi(u\beta)$ for all $u \in A^*$, as required.

Why is ψ shift-deterministic? Take any $a \in A$ and $\beta, \gamma \in A^\omega$ with $\psi(\beta) = \psi(\gamma)$. We have to show that $\psi(a\beta) = \psi(a\gamma)$. Indeed, assume that

$$0 = \psi(\beta) - \psi(\gamma) = \sum_{w \in A^*} \left(\frac{1}{|A|+1}\right)^{|w|} \big[\varphi(w\beta) - \varphi(w\gamma)\big].$$

If this series contains a non-zero term, then it must contain a positive term and a negative term. But this contradicts prefix-monotonicity of φ. So all the terms in this series must be 0. The same then must hold for a series:

$$\psi(a\beta) - \psi(a\gamma) = \sum_{w \in A^*} \left(\frac{1}{|A|+1}\right)^{|w|} \big[\varphi(wa\beta) - \varphi(wa\gamma)\big]$$

(all the terms in this series also appear in the series for $\psi(\beta) - \psi(\gamma)$). So we must have $\psi(a\beta) = \psi(a\gamma)$.

Why $\varphi = g \circ \psi$ for some continuous non-decreasing $g \colon \psi(A^\omega) \to \mathbb{R}$? Let us first show that

$$\varphi(\alpha) > \varphi(\beta) \implies \psi(\alpha) > \psi(\beta) \text{ for all } \alpha, \beta \in A^\omega. \tag{6}$$

Indeed, if $\varphi(\alpha) > \varphi(\beta)$, then we also have $\varphi(w\alpha) \geq \varphi(w\beta)$ for every $w \in A^*$, by prefix-monotonicity of φ. Now, by definition,

$$\psi(\alpha) - \psi(\beta) = \sum_{w \in A^*} \left(\frac{1}{|A|+1}\right)^{|w|} \big[\varphi(w\alpha) - \varphi(w\beta)\big].$$

All the terms in this series are non-negative, and the term corresponding to the empty w is strictly positive. So we have $\psi(\alpha) > \psi(\beta)$, as required.

Now, let us demonstrate that (6) implies that $\varphi = g \circ \psi$ for some non-decreasing $g \colon \psi(A^\omega) \to \mathbb{R}$. Namely, define g as follows. For $x \in \psi(A^\omega)$ take an arbitrary $\gamma \in \psi^{-1}(x)$ and set $g(x) = \varphi(\gamma)$. First, why do we have $\varphi = g \circ \psi$? By definition, $g(\psi(\alpha)) = \varphi(\gamma)$ for some $\gamma \in A^\omega$ with $\psi(\alpha) = \psi(\gamma)$. By (6) we also have $\varphi(\alpha) = \varphi(\gamma)$, so $g(\psi(\alpha)) = \varphi(\gamma) = \varphi(\alpha)$, as required. Now, why is g non-decreasing? I.e., why for all $x, y \in \psi(A^\omega)$ we have $x \leq y \implies g(x) \leq g(y)$? Indeed, $g(x) = \varphi(\gamma_x), g(y) = \varphi(\gamma_y)$ for some $\gamma_x \in \psi^{-1}(x)$ and $\gamma_y \in \psi^{-1}(y)$. Now, since $x \leq y$, we have $x = \psi(\gamma_x) \leq \psi(\gamma_y) = y$. By taking the contraposition of (6) we get that $g(x) = \varphi(\gamma_x) \leq \varphi(\gamma_y) = g(y)$, as required.

Finally, we show that any $g \colon \psi(A^\omega) \to \mathbb{R}$ with $\varphi = g \circ \psi$ must be continuous. For that we show that $|g(x) - g(y)| \leq |x - y|$ for all $x, y \in \psi(A^\omega)$. Take any $\alpha, \beta \in A^\omega$ with $x = \psi(\alpha)$ and $y = \psi(\beta)$. By prefix-monotonicity of φ we have that either $\varphi(w\alpha) \geq \varphi(w\beta)$ for all $w \in A^*$ or $\varphi(w\alpha) \leq \varphi(w\beta)$ for all $w \in A^*$. Up to swapping x and y we may assume that the first option holds. Then

$$\psi(\alpha) - \psi(\beta) = \sum_{w \in A^*} \left(\frac{1}{|A|+1}\right)^{|w|} \big[\varphi(w\alpha) - \varphi(w\beta)\big] \geq \varphi(\alpha) - \varphi(\beta) \geq 0.$$

On the left here we have $x - y$, and on the right we have $\varphi(\alpha) - \varphi(\beta) = g \circ \psi(\alpha) - g \circ \psi(\beta) = g(x) - g(y)$.

Transience in Countable MDPs

Stefan Kiefer
Department of Computer Science, University of Oxford, UK

Richard Mayr
School of Informatics, University of Edinburgh, UK

Mahsa Shirmohammadi
Université de Paris, CNRS, IRIF, F-75013 Paris, France

Patrick Totzke
Department of Computer Science, University of Liverpool, UK

Abstract

The `Transience` objective is not to visit any state infinitely often. While this is not possible in any finite Markov Decision Process (MDP), it can be satisfied in countably infinite ones, e.g., if the transition graph is acyclic.

We prove the following fundamental properties of `Transience` in countably infinite MDPs.

1. There exist uniformly ε-optimal MD strategies (memoryless deterministic) for `Transience`, even in infinitely branching MDPs.
2. Optimal strategies for `Transience` need not exist, even if the MDP is finitely branching. However, if an optimal strategy exists then there is also an optimal MD strategy.
3. If an MDP is universally transient (i.e., almost surely transient under all strategies) then many other objectives have a lower strategy complexity than in general MDPs. E.g., ε-optimal strategies for Safety and co-Büchi and optimal strategies for $\{0,1,2\}$-`Parity` (where they exist) can be chosen MD, even if the MDP is infinitely branching.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains; Mathematics of computing → Probability and statistics

Keywords and phrases Markov decision processes, Parity, Transience

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.11

Related Version *Full Version*: `https://arxiv.org/abs/2012.13739`

1 Introduction

> Those who cannot remember the past
> are condemned to repeat it.
>
> *George Santayana (1905) [22]*

The famous aphorism above has often been cited (with small variations), e.g., by Winston Churchill in a 1948 speech to the House of Commons, and carved into several monuments all over the world [22].

We prove that the aphorism is false. In fact, even those who cannot remember anything at all are *not* condemned to repeat the past. With the right strategy they can avoid repeating the past equally well as everyone else. More formally, playing for `Transience` does not require any memory. We show that there always exist ε-optimal memoryless deterministic strategies for `Transience`, and if optimal strategies exist then there also exist optimal memoryless deterministic strategies.[1]

[1] Our result applies to MDPs (also called games against nature). It is an open question whether it generalizes to countable stochastic 2-player games. (However, it is easy to see that the adversary needs infinite memory in general, even if the player is passive [14, 16].)

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 11; pp. 11:1–11:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Background. We study Markov decision processes (MDPs), a standard model for dynamic systems that exhibit both stochastic and controlled behavior [21]. MDPs play a prominent role in many domains, e.g., artificial intelligence and machine learning [26, 24], control theory [5, 1], operations research and finance [25, 12, 6, 23], and formal verification [2, 25, 11, 8, 3, 7].

An MDP is a directed graph where states are either random or controlled. Its observed behavior is described by runs, which are infinite paths that are, in part, determined by the choices of a controller. If the current state is random then the next state is chosen according to a fixed probability distribution. Otherwise, if the current state is controlled, the controller can choose a distribution over all possible successor states. By fixing a strategy for the controller (and initial state), one obtains a probability space of runs of the MDP. The goal of the controller is to optimize the expected value of some objective function on the runs.

The *strategy complexity* of a given objective characterizes the type of strategy necessary to achieve an optimal (resp. ε-optimal) value for the objective. General strategies can take the whole history of the run into account (history-dependent; (H)), while others use only bounded information about it (finite memory; (F)) or base decisions only on the current state (memoryless; (M)). Moreover, the strategy type depends on whether the controller can randomize (R) or is limited to deterministic choices (D). The simplest type, MD, refers to memoryless deterministic strategies.

Acyclicity and Transience. An MDP is called acyclic iff its transition graph is acyclic. While finite MDPs cannot be acyclic (unless they have deadlocks), countable MDPs can. In acyclic countable MDPs, the strategy complexity of Büchi/Parity objectives is lower than in the general case: ε-optimal strategies for Büchi/Parity objectives require only one bit of memory in acyclic MDPs, while they require infinite memory (an unbounded step-counter, plus one bit) in general countable MDPs [14, 15].

The concept of *transience* can be seen as a generalization of acyclicity. In a Markov chain, a state s is called *transient* iff the probability of returning from s to s is < 1 (otherwise the state is called recurrent). This means that a transient state is almost surely visited only finitely often. The concept of transient/recurrent is naturally lifted from Markov chains to MDPs, where they depend on the chosen strategy.

We define the `Transience` objective as the set of runs that do not visit any state infinitely often. We call an MDP *universally transient* iff it almost-surely satisfies `Transience` under every strategy. Thus every acyclic MDP is universally transient, but not vice-versa; cf. Figure 1. In particular, universal transience does not just depend on the structure of the transition graph, but also on the transition probabilities. Universally transient MDPs have interesting properties. Many objectives (e.g., Safety, Büchi, co-Büchi) have a lower strategy complexity than in general MDPs; see below.

We also study the strategy complexity of the `Transience` objective itself, and how it interacts with other objectives, e.g., how to attain a Büchi objective in a transient way.

Our contributions.

1. We show that there exist uniformly ε-optimal MD strategies (memoryless deterministic) for `Transience`, even in infinitely branching MDPs. This is unusual, since (apart from reachability objectives) most other objectives require infinite memory if the MDP is infinitely branching, e.g., all objectives generalizing Safety [17].

   Our result is shown in several steps. First we show that there exist ε-optimal deterministic 1-bit strategies for `Transience`. Then we show how to dispense with the 1-bit memory and obtain ε-optimal MD strategies for `Transience`. Finally, we make these MD strategies uniform, i.e., independent of the start state.

2. We show that optimal strategies for `Transience` need not exist, even if the MDP is finitely branching. If they do exist then there are also MD optimal strategies. More generally, there exists a single MD strategy that is optimal from every state that allows optimal strategies for `Transience`.
3. If an MDP is universally transient (i.e., almost surely transient under all strategies) then many other objectives have a lower strategy complexity than in general MDPs, e.g., ε-optimal strategies for Safety and co-Büchi and optimal strategies for $\{0,1,2\}$-`Parity` (where they exist) can be chosen MD, even if the MDP is infinitely branching.

For our proofs we develop some technical results that are of independent interest. We generalize Ornstein's plastering construction [20] from reachability to tail objectives and thus obtain a general tool to infer uniformly ε-optimal MD strategies from non-uniform ones (cf. Theorem 7). Secondly, in Section 6 we develop the notion of the *conditioned MDP* (cf. [17]). For tail objectives, this allows to obtain uniformly ε-optimal MD strategies wrt. *multiplicative errors* from those with merely additive errors.

2 Preliminaries

A *probability distribution* over a countable set S is a function $f : S \to [0,1]$ with $\sum_{s\in S} f(s) = 1$. We write $\mathcal{D}(S)$ for the set of all probability distributions over S.

Markov Decision Processes. We define Markov decision processes (MDPs for short) over countably infinite state spaces as tuples $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ where S is the countable set of states partitioned into a set $S_\Box$ of *controlled states* and a set $S_\bigcirc$ of *random states*. The *transition relation* is $\longrightarrow \subseteq S \times S$, and $P : S_\bigcirc \to \mathcal{D}(S)$ is a *probability function*. We write $s{\longrightarrow}s'$ if $(s,s') \in \longrightarrow$, and refer to s' as a *successor* of s. We assume that every state has at least one successor. The probability function P assigns to each random state $s \in S_\bigcirc$ a probability distribution $P(s)$ over its set of successors. A *sink* is a subset $T \subseteq S$ closed under the $\longrightarrow$ relation.

An MDP is *acyclic* if the underlying graph $(S, \longrightarrow)$ is acyclic. It is *finitely branching* if every state has finitely many successors and *infinitely branching* otherwise. An MDP without controlled states ($S_\Box = \emptyset$) is a *Markov chain*.

Strategies and Probability Measures. A *run* ρ is an infinite sequence $s_0 s_1 \cdots$ of states such that $s_i{\longrightarrow}s_{i+1}$ for all $i \in \mathbb{N}$; a *partial run* is a finite prefix of a run. We write $\rho(i) = s_i$ and say that (partial) run $s_0 s_1 \cdots$ *visits* s if $s = s_i$ for some i. It *starts in* s if $s = s_0$.

A *strategy* is a function $\sigma : S^* S_\Box \to \mathcal{D}(S)$ that assigns to partial runs $\rho s \in S^* S_\Box$ a distribution over the successors of s. We write $\Sigma_\mathcal{M}$ for the set of all strategies in $\mathcal{M}$. A strategy σ and an initial state $s_0 \in S$ induce a standard probability measure on sets of infinite runs. We write $\mathcal{P}_{\mathcal{M},s_0,\sigma}(\mathfrak{R})$ for the probability of a measurable set $\mathfrak{R} \subseteq s_0 S^\omega$ of runs starting from s_0. It is defined for the cylinders $s_0 s_1 \ldots s_n S^\omega \in S^\omega$ as $\mathcal{P}_{\mathcal{M},s_0,\sigma}(s_0 s_1 \ldots s_n S^\omega) \overset{\text{def}}{=} \prod_{i=0}^{n-1} \bar{\sigma}(s_0 s_1 \ldots s_i)(s_{i+1})$, where $\bar{\sigma}$ is the map that extends σ by $\bar{\sigma}(ws) = P(s)$ for all $ws \in S^* S_\bigcirc$. By Carathéodory's theorem [4], the measure for cylinders extends uniquely to a probability measure $\mathcal{P}_{\mathcal{M},s_0,\sigma}$ on all measurable subsets of $s_0 S^\omega$. We will write $\mathcal{E}_{\mathcal{M},s_0,\sigma}$ for the expectation w.r.t. $\mathcal{P}_{\mathcal{M},s_0,\sigma}$.

Strategy Classes. Strategies $\sigma : S^* S_\Box \to \mathcal{D}(S)$ are in general *randomized* (R) in the sense that they take values in $\mathcal{D}(S)$. A strategy σ is *deterministic* (D) if $\sigma(\rho)$ is a Dirac distribution for all partial runs $\rho \in S^* S_\Box$.

A formal definition of the amount of *memory* needed to implement strategies can be found in the full version [13]. The two classes of *memoryless* and *1-bit* strategies are central to this paper. A strategy σ is *memoryless* (M) if σ bases its decision only on the last state of the run: $\sigma(\rho s) = \sigma(\rho' s)$ for all $\rho, \rho' \in S^*$. We may view M-strategies as functions $\sigma : S_\Box \to \mathcal{D}(S)$. A 1-bit strategy σ may base its decision also on a memory mode $\mathsf{m} \in \{0,1\}$. Formally, a 1-bit strategy σ is given as a tuple (u, m_0) where $\mathsf{m}_0 \in \{0,1\}$ is the initial memory mode and $u : \{0,1\} \times S \to \mathcal{D}(\{0,1\} \times S)$ is an update function such that

- for all controlled states $s \in S_\Box$, the distribution $u((\mathsf{m}, s))$ is over $\{0,1\} \times \{s' \mid s {\longrightarrow} s'\}$.
- for all random states $s \in S_\bigcirc$, we have that $\sum_{\mathsf{m}' \in \{0,1\}} u((\mathsf{m}, s))(\mathsf{m}', s') = P(s)(s')$.

Note that this definition allows for updating the memory mode upon visiting random states. We write $\sigma[\mathsf{m}_0]$ for the strategy obtained from σ by setting the initial memory mode to m_0.

MD strategies are both memoryless and deterministic; and *deterministic 1-bit* strategies are both deterministic and 1-bit.

Objectives. The objective of the controller is determined by a predicate on infinite runs. We assume familiarity with the syntax and semantics of the temporal logic LTL [9]. Formulas are interpreted on the underlying structure $(S, \longrightarrow)$ of the MDP $\mathcal{M}$. We use $[\![\varphi]\!]^{\mathcal{M},s} \subseteq sS^\omega$ to denote the set of runs starting from s that satisfy the LTL formula φ, which is a measurable set [27]. We also write $[\![\varphi]\!]^{\mathcal{M}}$ for $\bigcup_{s \in S} [\![\varphi]\!]^{\mathcal{M},s}$. Where it does not cause confusion we will identify φ and $[\![\varphi]\!]$ and just write $\mathcal{P}_{\mathcal{M},s,\sigma}(\varphi)$ instead of $\mathcal{P}_{\mathcal{M},s,\sigma}([\![\varphi]\!]^{\mathcal{M},s})$.

Given a set $T \subseteq S$ of states, the *reachability* objective $\texttt{Reach}(T) \overset{\text{def}}{=} \mathsf{F}T$ is the set of runs that visit T at least once. The *safety* objective $\texttt{Safety}(T) \overset{\text{def}}{=} \mathsf{G}\neg T$ is the set of runs that never visit T.

Let $\mathcal{C} \subseteq \mathbb{N}$ be a finite set of colors. A *color function* $Col : S \to \mathcal{C}$ assigns to each state s its color $Col(s)$. The parity objective, written as $\texttt{Parity}(Col)$, is the set of infinite runs such that the largest color that occurs infinitely often along the run is even. To define this formally, let $even(\mathcal{C}) = \{i \in \mathcal{C} \mid i \equiv 0 \mod 2\}$. For $\rhd \in \{<, \leq, =, \geq, >\}$, $n \in \mathbb{N}$, and $Q \subseteq S$, let $[Q]^{Col \rhd n} \overset{\text{def}}{=} \{s \in Q \mid Col(s) \rhd n\}$ be the set of states in Q with color $\rhd n$. Then

$$\texttt{Parity}(Col) \overset{\text{def}}{=} \bigvee_{i \in even(\mathcal{C})} \left(\mathsf{GF}[S]^{Col=i} \wedge \mathsf{FG}[S]^{Col \leq i} \right).$$

We write $\mathcal{C}$-$\texttt{Parity}$ for the parity objectives with the set of colors $\mathcal{C} \subseteq \mathbb{N}$. The classical Büchi and co-Büchi objectives correspond to $\{1,2\}$-$\texttt{Parity}$ and $\{0,1\}$-$\texttt{Parity}$, respectively.

An objective φ is called a *tail objective* (in $\mathcal{M}$) iff for every run $\rho'\rho$ with some finite prefix ρ' we have $\rho'\rho \in \varphi \Leftrightarrow \rho \in \varphi$. For every coloring Col, $\texttt{Parity}(Col)$ is tail. Reachability objectives are not always tail but in MDPs where the target set T is a sink $\texttt{Reach}(T)$ is tail.

Optimal and ε-optimal Strategies. Given an objective φ, the *value* of state s in an MDP $\mathcal{M}$, denoted by $\texttt{val}_{\mathcal{M},\varphi}(s)$, is the supremum probability of achieving φ. Formally, we have $\texttt{val}_{\mathcal{M},\varphi}(s) \overset{\text{def}}{=} \sup_{\sigma \in \Sigma} \mathcal{P}_{\mathcal{M},s,\sigma}(\varphi)$ where Σ is the set of all strategies. For $\varepsilon \geq 0$ and state $s \in S$, we say that a strategy is *ε-optimal* from s iff $\mathcal{P}_{\mathcal{M},s,\sigma}(\varphi) \geq \texttt{val}_{\mathcal{M},\varphi}(s) - \varepsilon$. A 0-optimal strategy is called *optimal.* An optimal strategy is *almost-surely winning* iff $\texttt{val}_{\mathcal{M},\varphi}(s) = 1$.

Considering an MD strategy as a function $\sigma : S_\Box \to S$ and $\varepsilon \geq 0$, σ is *uniformly* ε-optimal (resp. uniformly optimal) if it is ε-optimal (resp. optimal) from every $s \in S$.

Throughout the paper, we may drop the subscripts and superscripts from notations, if it is understood from the context. The missing proofs can be found in the full version [13].

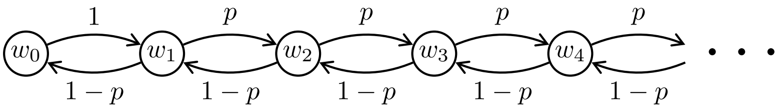


Figure 1 Gambler's Ruin with restart: The state w_i illustrates that the controller's wealth is i, and the coin tosses are in the controller's favor with probability p. For all i, $\mathcal{P}_{w_i}(\texttt{Transience}) = 0$ if $p \leq \frac{1}{2}$; and $\mathcal{P}_{w_i}(\texttt{Transience}) = 1$ otherwise.

3 Transience and Universally Transient MDPs

In this section we define the transience property for MDPs, a natural generalization of the well-understood concept of transient Markov chains. We enumerate crucial characteristics of this objective and define the notion of universally transient MDPs.

Fix a countable MDP $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$. Define the transience objective, denoted by Transience, to be the set of runs that do not visit any state of $\mathcal{M}$ infinitely often, i.e.,

$$\texttt{Transience} \overset{\text{def}}{=} \bigwedge_{s \in S} \mathsf{FG}\, \neg s.$$

The Transience objective is tail, as it is closed under removing finite prefixes of runs. Also note that Transience cannot be encoded in a parity objective.

We call $\mathcal{M}$ *universally transient* iff for all states s_0, for all strategies σ, the Transience property holds almost-surely from s_0, i.e.,

$$\forall s_0 \in S \;\; \forall \sigma \in \Sigma \;\; \mathcal{P}_{\mathcal{M},s_0,\sigma}(\texttt{Transience}) = 1.$$

The MDP in Figure 1 models the classical Gambler's Ruin Problem with restart; see [10, Chapter 14]. It is well-known that if the controller starts with wealth i and if $p \leq \frac{1}{2}$, the probability of ruin (visiting the state w_0) is $\mathcal{P}_{w_i}(\mathsf{F}\, w_0) = 1$. Consequently, the probability of re-visiting w_0 infinitely often is 1, implying that $\mathcal{P}_{w_i}(\texttt{Transience}) = 0$. In contrast, for the case with $p > \frac{1}{2}$, for all states w_i, the probability of re-visiting w_i is strictly below 1. Hence, the Transience property holds almost-surely. This example indicates that the transience property depends on the probability values of the transitions and not just on the underlying transition graph, and thus may require arithmetic reasoning. In particular, the MDP in Figure 1 is universally transient iff $p > \frac{1}{2}$.

In general, optimal strategies for Transience need not exist:

▶ **Lemma 1.** *There exists a finitely branching countable MDP with initial state s_0 such that*

- $\texttt{val}_{\texttt{Transience}}(s) = 1$ *for all controlled states s,*
- *there does not exist any optimal strategy σ such that $\mathcal{P}_{s_0,\sigma}(\texttt{Transience}) = 1$.*

Proof. Consider a countable MDP $\mathcal{M}$ with set $S = \{\ell_i, \ell'_i, r_i, x_i \mid i \geq 1\} \cup \{\ell_0, \bot\}$ of states; see Figure 2. For all $i \geq 1$ the state x_{i+1} is the unique successor of x_i so that $(x_i)_{i\geq 1}$ form an acyclic ladder; the value of Transience is 1 for all x_i. The state $\bot$ is sink, and its value is 0. The states $(r_i)_{i\geq 1}$ are all random, and $r_i \xrightarrow{1-2^{-i}} x_i$ and $r_i \xrightarrow{2^{-i}} \bot$. Observe that the value of Transience is $1 - 2^{-i}$ for the r_i.

The states $(\ell_i)_{i\in\mathbb{N}}$ are controlled whereas the states $(\ell'_i)_{i\geq 1}$ are random. By interleaving of these states, we construct a "recurrent ladder" of decisions: $\ell_0 \to \ell_1$ and for all $i \geq 1$, state ℓ_i has two successors ℓ'_i and r_i. In random states ℓ'_i, as in Gambler's Ruin with a fair coin, the successors are ℓ_{i-1} or ℓ_{i+1}, each with equal probability. In each state $(\ell_i)_{i\geq 1}$, the controller decides to either stay on the ladder by going to ℓ'_i or leaves the ladder to r_i. As in Figure 1, if the controller stays on the ladder forever, the probability of Transience is 0.

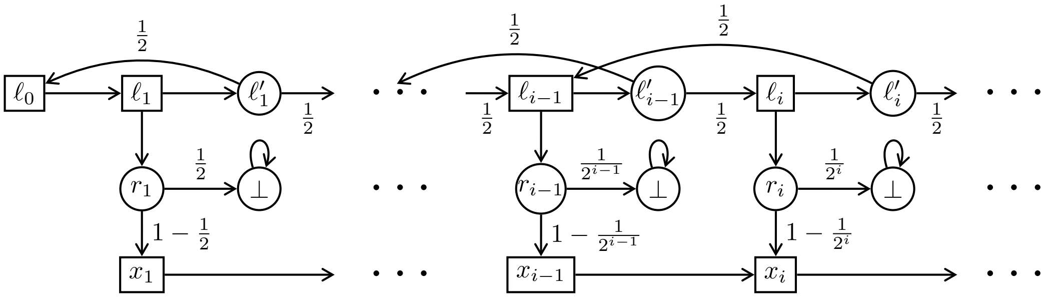


Figure 2 A partial illustration of the MDP in Lemma 1, in which there is no optimal strategy for `Transience`, starting from states ℓ_i. For readability, we have three copies of the state $\perp$. We call the ladder consisting of the interleaved controlled states ℓ_i and random states ℓ_i' a "recurrent ladder": if the controller stays on this ladder forever, it faithfully simulates a Gambler's Ruin with a fair coin, and the probability of `Transience` will be 0.

Starting in ℓ_0, for all $i > 0$, strategy σ_i that stays on the ladder until visiting ℓ_i (which happens eventually almost surely) and then leaves the ladder to r_i achieves `Transience` with probability $1 - 2^i$. Hence, $\mathtt{val}_{\mathtt{Transience}}(\ell_0) = 1$.

Recall that transience cannot be achieved with a positive probability by staying on the acyclic ladder forever. But any strategy that leaves the ladder with a positive probability comes with a positive probability of falling into $\perp$, thus is not optimal either. Thus there is no optimal strategy for `Transience`. ◀

Reduction to Finitely Branching MDPs. In our main results, we will prove that for the `Transience` property there always exist ε-optimal MD strategies in finitely branching countable MDPs; and if an optimal strategy exists, there will exist an optimal MD strategy. We generalize these results to infinitely branching countable MDPs by the following reduction:

▶ **Lemma 2.** *Given an infinitely branching countable MDP $\mathcal{M}$ with an initial state s_0, there exists a finitely branching countable $\mathcal{M}'$ with a set S' of states such that $s_0 \in S'$ and*

1. *each strategy α_1 in $\mathcal{M}$ is mapped to a unique strategy β_1 in $\mathcal{M}'$ where*

$$\mathcal{P}_{s_0,\alpha_1}(\mathtt{Transience}) = \mathcal{P}_{s_0,\beta_1}(\mathtt{Transience}),$$

2. *and conversely, every MD strategy β_2 in $\mathcal{M}'$ is mapped to an MD strategy α_2 in $\mathcal{M}$ where*

$$\mathcal{P}_{s_0,\alpha_2}(\mathtt{Transience}) \geq \mathcal{P}_{s_0,\beta_2}(\mathtt{Transience}).$$

Properties of Universally Transient MDPs. Notice that acyclicity implies universal transience, but not vice-versa.

▶ **Lemma 3.** *For every countable MDP $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$, the following conditions are equivalent.*

1. *$\mathcal{M}$ is universally transient, i.e., $\forall s_0, \forall \sigma. \mathcal{P}_{\mathcal{M},s_0,\sigma}(\mathtt{Transience}) = 1$.*
2. *For every initial state s_0 and state s, the objective of re-visiting s infinitely often has value zero, i.e., $\forall s_0, s \sup_\sigma \mathcal{P}_{\mathcal{M},s_0,\sigma}(\mathsf{GF}(s)) = 0$.*
3. *For every state s the value of the objective to re-visit s is strictly below 1, i.e., $Re(s) \overset{def}{=} \sup_\sigma \mathcal{P}_{\mathcal{M},s,\sigma}(\mathsf{XF}(s)) < 1$.*

4. *For every state s there exists a finite bound $B(s)$ such that for every state s_0 and strategy σ from s_0 the expected number of visits to s is $\leq B(s)$.*
5. *For all states s_0, s, under every strategy σ from s_0 the expected number of visits to s is finite.*

Proof. Towards (1) ⇒ (2), consider an arbitrary strategy σ from the initial state s_0 and some state s. By (1) we have $\forall\sigma.\mathcal{P}_{\mathcal{M},s_0,\sigma}(\texttt{Transience}) = 1$ and thus $0 = \mathcal{P}_{\mathcal{M},s_0,\sigma}(\neg\texttt{Transience}) = \mathcal{P}_{\mathcal{M},s_0,\sigma}(\bigcup_{s'\in S}\mathsf{GF}(s')) \geq \mathcal{P}_{\mathcal{M},s_0,\sigma}(\mathsf{GF}(s))$ which implies (2).

Towards (2) ⇒ (1), consider an arbitrary strategy σ from the initial state s_0. By (2) we have $0 = \sum_{s\in S}\mathcal{P}_{\mathcal{M},s_0,\sigma}(\mathsf{GF}(s)) \geq \mathcal{P}_{\mathcal{M},s_0,\sigma}(\bigcup_{s\in S}\mathsf{GF}(s)) = \mathcal{P}_{\mathcal{M},s_0,\sigma}(\neg\texttt{Transience})$ and thus $\mathcal{P}_{\mathcal{M},s_0,\sigma}(\texttt{Transience}) = 1$.

We now show the implications (2) ⇒ (3) ⇒ (4) ⇒ (5) ⇒ (2).

Towards ¬(3) ⇒ ¬(2), ¬(3) implies $\exists s.\mathit{Re}(s) = 1$ and thus $\forall\varepsilon > 0.\exists\sigma_\varepsilon\,\mathcal{P}_{\mathcal{M},s,\sigma_\varepsilon}(\mathsf{XF}(s)) \geq 1-\varepsilon$. Let $\varepsilon_i \overset{\text{def}}{=} 2^{-(i+1)}$. We define the strategy σ to play like σ_{ε_i} between the i-th and $(i+1)$th visit to s. Since $\sum_{i=1}^{\infty}\varepsilon_i < \infty$, we have $\prod_{i=1}^{\infty}(1-\varepsilon_i) > 0$. Therefore $\mathcal{P}_{\mathcal{M},s,\sigma}(\mathsf{GF}(s)) \geq \prod_{i=1}^{\infty}(1-\varepsilon_i) > 0$, which implies ¬(2), where $s_0 = s$.

Towards (3) ⇒ (4), regardless of s_0 and the chosen strategy, the expected number of visits to s is upper-bounded by $B(s) \overset{\text{def}}{=} \sum_{n=0}^{\infty}(n+1)\cdot(\mathit{Re}(s))^n < \infty$.

The implication (4) ⇒ (5) holds trivially.

Towards ¬(2) ⇒ ¬(5), by ¬(2) there exist states s_0, s and a strategy σ such that $\mathcal{P}_{\mathcal{M},s_0,\sigma}(\mathsf{GF}(s)) > 0$. Thus the expected number of visits to s is infinite, which implies ¬(5). ◀

We remark that if an MDP is *not* universally transient (unlike in Lemma 3(5)), for a strategy σ, the expected number of visits to some state can be infinite, even if σ attains `Transience` almost surely.

Consider the MDP $\mathcal{M}$ with controlled states $\{s_0, s_1, \dots\}$, initial state s_0 and transitions $s_0 \to s_0$ and $s_k \to s_{k+1}$ for every $k \geq 0$. We define a strategy σ that, while in state s_0, proceeds in rounds $i = 1, 2, \dots$. In the i-th round it tosses a fair coin. If Heads then it goes to s_1. If Tails then it loops around s_0 exactly 2^i times and then goes to round $i+1$. In every round the probability of going to s_1 is $1/2$ and therefore the probability of staying in s_0 forever is $(1/2)^\infty = 0$. Thus $\mathcal{P}_{\mathcal{M},s_0,\sigma}(\texttt{Transience}) = 1$. However, the expected number of visits to s_0 is $\geq \sum_{i=1}^{\infty}\left(\frac{1}{2}\right)^i \cdot 2^i = \infty$.

4 MD Strategies for Transience

We show that there exist uniformly ε-optimal MD strategies for `Transience` and that optimal strategies, where they exist, can also be chosen MD.

First we show that there exist ε-optimal deterministic 1-bit strategies for `Transience` (in Corollary 5) and then we show how to dispense with the 1-bit memory (in Lemma 6).

It was shown in [14] that there exist ε-optimal deterministic 1-bit strategies for Büchi objectives in *acyclic* countable MDPs (though not in general MDPs). These 1-bit strategies will be similar to the 1-bit strategies for `Transience` that we aim for in (not necessarily acyclic) countable MDPs. In Lemma 4 below we first strengthen the result from [14] and construct ε-optimal deterministic 1-bit strategies for objectives $\texttt{Büchi}(F) \cap \texttt{Transience}$. From this we obtain deterministic 1-bit strategies for `Transience` (Corollary 5).

▶ **Lemma 4.** *Let $\mathcal{M}$ be a countable MDP, I a finite set of initial states, F a set of states and $\varepsilon > 0$. Then there exists a deterministic 1-bit strategy for $\texttt{Büchi}(F) \cap \texttt{Transience}$ that is ε-optimal from every $s \in I$.*

Proof sketch. It follows the proof of [14, Theorem 5], which considers $\texttt{Büchi}(F)$ conditions for *acyclic* (and hence universally transient) MDPs. The only part of that proof that requires modification is [14, Lemma 10], which is replaced here by [13, Lemma 18] to deal with general MDPs.

In short, from every $s \in I$ there exists an ε-optimal strategy σ_s for $\varphi \overset{\text{def}}{=} \texttt{Büchi}(F) \cap \texttt{Transience}$. We observe the behavior of the finitely many σ_s for $s \in I$ on an infinite, increasing sequence of finite subsets of S. Based on [13, Lemma 18], we can define a second stronger objective $\varphi' \subseteq \varphi$ and show $\forall_{s \in I}\, \mathcal{P}_{\mathcal{M},s,\sigma_s}(\varphi') \geq \texttt{val}_{\mathcal{M},\varphi}(s) - 2\varepsilon$. We then construct a deterministic 1-bit strategy σ' that is optimal for φ' from all $s \in I$ and thus 2ε-optimal for φ. Since ε can be chosen arbitrarily small, the result follows. ◀

Unlike for the $\texttt{Transience}$ objective alone (see below), the 1-bit memory is strictly necessary for the $\texttt{Büchi}(F) \cap \texttt{Transience}$ objective in Lemma 4. The 1-bit lower bound for $\texttt{Büchi}(F)$ objectives in [14] holds even for acyclic MDPs where $\texttt{Transience}$ is trivially true.

▶ **Corollary 5.** *Let $\mathcal{M}$ be a countable MDP, I a finite set of initial states, F a set of states and $\varepsilon > 0$.*

1. *If $\forall s \in I$ $\texttt{val}_{\mathcal{M},\texttt{Büchi}(F)}(s) = \texttt{val}_{\mathcal{M},\texttt{Büchi}(F)\cap\texttt{Transience}}(s)$ then there exists a deterministic 1-bit strategy for $\texttt{Büchi}(F)$ that is ε-optimal from every $s \in I$.*
2. *If $\mathcal{M}$ is universally transient then there exists a deterministic 1-bit strategy for $\texttt{Büchi}(F)$ that is ε-optimal from every $s \in I$.*
3. *There exists a deterministic 1-bit strategy for $\texttt{Transience}$ that is ε-optimal from every $s \in I$.*

Proof. Towards (1), since $\forall s \in I$ $\texttt{val}_{\mathcal{M},\texttt{Büchi}(F)}(s) = \texttt{val}_{\mathcal{M},\texttt{Büchi}(F)\cap\texttt{Transience}}(s)$, strategies that are ε-optimal for $\texttt{Büchi}(F) \cap \texttt{Transience}$ are also ε-optimal for $\texttt{Büchi}(F)$. Thus the result follows from Lemma 4.

Item (2) follows directly from (1), since the precondition always holds in universally transient MDPs.

Towards (3), let $F \overset{\text{def}}{=} S$. Then we have $\texttt{Büchi}(F) \cap \texttt{Transience} = \texttt{Transience}$ and we obtain from Lemma 4 that there exists a deterministic 1-bit strategy for $\texttt{Transience}$ that is ε-optimal from every $s \in I$. ◀

Note that every acyclic MDP is universally transient and thus Corollary 5(2) implies the upper bound on the strategy complexity of $\texttt{Büchi}(F)$ from [14] (but not vice-versa).

In the next step we show how to dispense with the 1-bit memory and obtain non-uniform ε-optimal MD strategies for $\texttt{Transience}$.

▶ **Lemma 6.** *Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ be a countable MDP with initial state s_0, and $\varepsilon > 0$. There exists an MD strategy σ that is ε-optimal for $\texttt{Transience}$ from s_0, i.e., $\mathcal{P}_{\mathcal{M},s_0,\sigma}(\texttt{Transience}) \geq \texttt{val}_{\mathcal{M},\texttt{Transience}}(s_0) - \varepsilon$.*

Proof. By Lemma 2 it suffices to prove the property for finitely branching MDPs. Thus without restriction in the rest of the proof we assume that $\mathcal{M}$ is finitely branching.

Let $\varepsilon' \overset{\text{def}}{=} \varepsilon/2$. We instantiate Corollary 5(3) with $I \overset{\text{def}}{=} \{s_0\}$ and obtain that there exists an ε'-optimal deterministic 1-bit strategy $\hat{\sigma}$ for $\texttt{Transience}$ from s_0.

We now construct a slightly modified MDP $\mathcal{M}'$ as follows. Let $S_{bad} \subseteq S$ be the subset of states where $\hat{\sigma}$ attains zero for $\texttt{Transience}$ in *both* memory modes, i.e., $S_{bad} \overset{\text{def}}{=} \{s \in S \mid \mathcal{P}_{\mathcal{M},s,\sigma[0]}(\texttt{Transience}) = \mathcal{P}_{\mathcal{M},s,\sigma[1]}(\texttt{Transience}) = 0\}$. Let $S_{good} \overset{\text{def}}{=} S \setminus S_{bad}$. We obtain $\mathcal{M}'$ from $\mathcal{M}$ by making all states in S_{bad} losing sinks (for $\texttt{Transience}$), by deleting all outgoing edges and adding a self-loop instead. It follows that

$$\mathcal{P}_{\mathcal{M},s_0,\hat{\sigma}}(\mathtt{Transience}) = \mathcal{P}_{\mathcal{M}',s_0,\hat{\sigma}}(\mathtt{Transience}) \tag{1}$$

$$\forall \sigma.\ \mathcal{P}_{\mathcal{M},s_0,\sigma}(\mathtt{Transience}) \geq \mathcal{P}_{\mathcal{M}',s_0,\sigma}(\mathtt{Transience}) \tag{2}$$

In the following we show that it is possible to play in such a way that, for every $s \in S_{good}$, the expected number of visits to s is *finite*. We obtain the deterministic 1-bit strategy σ' in $\mathcal{M}'$ by modifying $\hat{\sigma}$ as follows. In every state s and memory mode $x \in \{0,1\}$ where $\hat{\sigma}[x]$ attains 0 for Transience and $\hat{\sigma}[1-x]$ attains > 0 the strategy σ' sets the memory bit to $1-x$. (Note that only states $s \in S_{good}$ can be affected by this change.) It follows that

$$\forall s \in S.\ \mathcal{P}_{\mathcal{M}',s,\sigma'}(\mathtt{Transience}) \geq \mathcal{P}_{\mathcal{M}',s,\hat{\sigma}}(\mathtt{Transience}) \tag{3}$$

Moreover, from all states in S_{good} in $\mathcal{M}'$ the strategy σ' attains a strictly positive probability of Transience in *both* memory modes, i.e., for all $s \in S_{good}$ we have

$$t(s,\sigma') \overset{\text{def}}{=} \min_{x\in\{0,1\}} \mathcal{P}_{\mathcal{M}',s,\sigma'[i]}(\mathtt{Transience}) > 0.$$

Let $r(s,\sigma',x)$ be the probability, when playing $\sigma'[x]$ from state s, of reaching s again in the *same* memory mode x. For every $s \in S_{good}$ we have $r(s,\sigma',x) < 1$, since $t(s,\sigma') > 0$.

Let $R(s)$ be the expected number of visits to state s when playing σ' from s_0 in $\mathcal{M}'$, and $R_x(s)$ the expected number of visits to s in memory mode $x \in \{0,1\}$. For all $s \in S_{good}$ we have that

$$R(s) = R_0(s) + R_1(s) \leq \sum_{n=1}^{\infty} n \cdot r(s,\sigma',0)^{n-1} + \sum_{n=1}^{\infty} n \cdot r(s,\sigma',1)^{n-1} < \infty \tag{4}$$

where the first equality holds by linearity of expectations. Thus the expected number of visits to s is *finite*.

Now we upper-bound the probability of visiting S_{bad}. We have $\mathcal{P}_{\mathcal{M}',s_0,\sigma'}(\mathtt{Transience}) \geq \mathcal{P}_{\mathcal{M}',s_0,\hat{\sigma}}(\mathtt{Transience}) = \mathcal{P}_{\mathcal{M},s_0,\hat{\sigma}}(\mathtt{Transience}) \geq \mathtt{val}_{\mathcal{M},\mathtt{Transience}}(s_0) - \varepsilon'$ by (3), (1) and the ε'-optimality of $\hat{\sigma}$. Since states in S_{bad} are losing sinks in $\mathcal{M}'$, it follows that

$$\mathcal{P}_{\mathcal{M}',s_0,\sigma'}(\mathsf{F} S_{bad}) \leq 1 - \mathcal{P}_{\mathcal{M}',s_0,\sigma'}(\mathtt{Transience}) \leq 1 - \mathtt{val}_{\mathcal{M},\mathtt{Transience}}(s_0) + \varepsilon' \tag{5}$$

We now augment the MDP $\mathcal{M}'$ by assigning costs to transitions as follows. Let $i : S \to \mathbb{N}$ be an enumeration of the state space, i.e., a bijection. Let $S'_{good} \overset{\text{def}}{=} \{s \in S_{good} \mid R(s) > 0\}$ be the subset of states in S_{good} that are visited with non-zero probability when playing σ' from s_0. Each transition $s' \to s$ is assigned a cost:

- If $s' \in S_{bad}$ then $s \in S_{bad}$ by def. of $\mathcal{M}'$. We assign cost 0.
- If $s' \in S_{good}$ and $s \in S_{bad}$ we assign cost $K/(1 - \mathtt{val}_{\mathcal{M},\mathtt{Transience}}(s_0) + \varepsilon')$ for $K \overset{\text{def}}{=} (1+\varepsilon')/\varepsilon'$.
- If $s' \in S_{good}$ and $s \in S'_{good}$ we assign cost $2^{-i(s)}/R(s)$. This is well defined, since $R(s) > 0$.
- $s' \in S_{good}$ and $s \in S_{good} \setminus S'_{good}$ we assign cost 1.

Note that all transitions leading to states in S_{good} are assigned a non-zero cost, since $R(s)$ is finite by (4).

When playing σ' from s_0 in $\mathcal{M}'$, the expected total cost is upper-bounded by

$$\mathcal{P}_{\mathcal{M}',s_0,\sigma'}(\mathsf{F} S_{bad}) \cdot K/(1 - \mathtt{val}_{\mathcal{M},\mathtt{Transience}}(s_0) + \varepsilon') + \sum_{s\in S'_{good}} R(s) \cdot 2^{-i(s)}/R(s)$$

The first part is $\leq K$ by (5) and the second part is ≤ 1, since $R(s) < \infty$ by (4). Therefore the expected total cost is $\leq K+1$, i.e., σ' witnesses that it is possible to attain a finite expected cost that is upper-bounded by $K+1$.

Now we define our MD strategy σ. Let σ be an optimal MD strategy on $\mathcal{M}'$ (from s_0) that minimizes the expected cost. It exists, as a finite expected cost is attainable and $\mathcal{M}'$ is finitely branching; see [21, Theorem 7.3.6].

We now show that σ attains `Transience` with high probability in $\mathcal{M}'$ (and in $\mathcal{M}$). Since σ is cost-optimal, its attained cost from s_0 is upper-bounded by that of σ', i.e., $\leq K+1$. Since the cost of entering S_{bad} is $K/(1-\mathtt{val}_{\mathcal{M},\mathtt{Transience}}(s_0)+\varepsilon')$, we have $\mathcal{P}_{\mathcal{M}',s_0,\sigma}(\mathsf{F}S_{bad})\cdot K/(1-\mathtt{val}_{\mathcal{M},\mathtt{Transience}}(s_0)+\varepsilon') \leq K+1$ and thus

$$\mathcal{P}_{\mathcal{M}',s_0,\sigma}(\mathsf{F}S_{bad}) \leq \frac{K+1}{K}(1-\mathtt{val}_{\mathcal{M},\mathtt{Transience}}(s_0)+\varepsilon') \tag{6}$$

For every state $s \in S_{good}$, all transitions into s have the same fixed non-zero cost. Thus every run that visits some state $s \in S_{good}$ infinitely often has infinite cost. Since the expected cost of playing σ from s_0 is $\leq K+1$, such runs must be a null-set, i.e.,

$$\mathcal{P}_{\mathcal{M}',s_0,\sigma}(\neg\mathtt{Transience} \wedge \mathsf{G}S_{good}) = 0 \tag{7}$$

Thus

$$\begin{aligned}
&\mathcal{P}_{\mathcal{M},s_0,\sigma}(\mathtt{Transience}) \\
&\geq \mathcal{P}_{\mathcal{M}',s_0,\sigma}(\mathtt{Transience}) && \text{by (2)}\\
&= 1-\mathcal{P}_{\mathcal{M}',s_0,\sigma}(\mathsf{F}S_{bad}) && \text{by (7)}\\
&\geq 1-\frac{K+1}{K}(1-\mathtt{val}_{\mathcal{M},\mathtt{Transience}}(s_0)+\varepsilon') && \text{by (6)}\\
&= \mathtt{val}_{\mathcal{M},\mathtt{Transience}}(s_0)-\varepsilon'-(1/K)(1-\mathtt{val}_{\mathcal{M},\mathtt{Transience}}(s_0)+\varepsilon')\\
&\geq \mathtt{val}_{\mathcal{M},\mathtt{Transience}}(s_0)-\varepsilon'-(1/K)(1+\varepsilon')\\
&= \mathtt{val}_{\mathcal{M},\mathtt{Transience}}(s_0)-2\varepsilon' && \text{def. of } K\\
&= \mathtt{val}_{\mathcal{M},\mathtt{Transience}}(s_0)-\varepsilon && \text{def. of } \varepsilon'
\end{aligned}$$ ◀

Now we lift the result of Lemma 6 from non-uniform to uniform strategies (and to optimal strategies) and obtain the following theorem. The proof is a generalization of a "plastering" construction by Ornstein [20] (see also [16]) from reachability to tail objectives, which works by fixing MD strategies on ever expanding subsets of the state space.

▶ **Theorem 7.** *Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ be a countable MDP, and let φ be an objective that is tail in $\mathcal{M}$. Suppose for every $s \in S$ there exist ε-optimal MD strategies for φ. Then:*

1. *There exist uniform ε-optimal MD strategies for φ.*
2. *There exists a single MD strategy that is optimal from every state that has an optimal strategy.*

▶ **Theorem 8.** *In every countable MDP there exist uniform ε-optimal MD strategies for* `Transience`*. Moreover, there exists a single MD strategy that is optimal for* `Transience` *from every state that has an optimal strategy.*

Proof. Immediate from Lemma 6 and Theorem 7, since `Transience` is a tail objective. ◀

5 Strategy Complexity in Universally Transient MDPs

The strategy complexity of parity objectives in general MDPs is known [15]. Here we show that some parity objectives have a lower strategy complexity in universally transient MDPs. It is known [14] that there are acyclic (and hence universally transient) MDPs where ε-optimal strategies for $\{1,2\}$-$\texttt{Parity}$ (and optimal strategies for $\{1,2,3\}$-$\texttt{Parity}$, resp.) require 1 bit.

We show that, for all simpler parity objectives in the Mostowski hierarchy [19], universally transient MDPs admit uniformly (ε-)optimal MD strategies (unlike general MDPs [15]). These results (Theorems 10 and 11) ultimately rely on the existence of uniformly ε-optimal strategies for safety objectives. While such strategies always exist for finitely branching MDPs – simply pick a value-maximal successor – this is not the case for infinitely branching MDPs [17]. However, we show that universal transience implies the existence of uniformly ε-optimal strategies for safety objectives even for *infinitely branching* MDPs.

▶ **Theorem 9.** *For every universally transient countable MDP, safety objective and $\varepsilon > 0$ there exists a uniformly ϵ-optimal MD strategy.*

Proof. Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ be a universally transient MDP and $\varepsilon > 0$. Assume w.l.o.g. that the target $T \subseteq S$ of the objective $\varphi = \texttt{Safety}(T)$ is a (losing) sink and let $\iota : S \to \mathbb{N}$ be an enumeration of the state space S.

By Lemma 3(3), for every state s we have $Re(s) \overset{\text{def}}{=} \sup_\sigma \mathcal{P}_{\mathcal{M},s,\sigma}(\mathsf{XF}(s)) < 1$ and thus $R(s) \overset{\text{def}}{=} \sum_{i=0}^{\infty} Re(s)^i < \infty$. This means that, independent of the chosen strategy, $Re(s)$ upper-bounds the chance to return to s, and $R(s)$ bounds the expected number of visits to s.

Suppose that σ is an MD strategy which, at any state $s \in S_\Box$, picks a successor s' with

$$\texttt{val}(s') \quad \geq \quad \texttt{val}(s) - \frac{\varepsilon}{2^{\iota(s)+1} \cdot R(s)}.$$

This is possible even if $\mathcal{M}$ is infinitely branching, by the definition of value and the fact that $R(s) < \infty$. We show that $\mathcal{P}_{\mathcal{M},s_0,\sigma}(\texttt{Safety}(T)) \geq \texttt{val}(s_0) - \varepsilon$ holds for every initial state s_0, which implies the claim of the theorem.

Towards this, we define a function $\texttt{cost}$ that labels each transition in the MDP with a real-valued cost: For every controlled transition $s \longrightarrow s'$ let $\texttt{cost}((s,s')) \overset{\text{def}}{=} \texttt{val}(s) - \texttt{val}(s') \geq 0$. Random transitions have cost zero. We will argue that when playing σ from any start state s_0, its attainment w.r.t. the objective $\texttt{Safety}(T)$ equals the value of s_0 minus the expected total cost, and that this cost is bounded by ε.

For any $i \in \mathbb{N}$ let us write s_i for the random variable denoting the state just after step i, and $\texttt{Cost}(i) \overset{\text{def}}{=} \texttt{cost}(s_i, s_{i+1})$ for the cost of step i in a random run. We observe that under σ the expected total cost is bounded in the limit, i.e.,

$$\lim_{n\to\infty} \mathcal{E}\left(\sum_{i=0}^{n-1} \texttt{Cost}(i)\right) \leq \varepsilon. \tag{8}$$

We moreover note that for every n,

$$\mathcal{E}(\texttt{val}(s_n)) = \mathcal{E}(\texttt{val}(s_0)) - \mathcal{E}\left(\sum_{i=0}^{n-1} \texttt{Cost}(i)\right). \tag{9}$$

Full proofs of the above two equations can be found in [13]. Together they imply

$$\liminf_{n\to\infty} \mathcal{E}(\texttt{val}(s_n)) = \texttt{val}(s_0) - \lim_{n\to\infty} \mathcal{E}\left(\sum_{i=0}^{n-1} \texttt{cost}(i)\right) \geq \texttt{val}(s_0) - \varepsilon. \tag{10}$$

Finally, to show the claim let $[s_n \notin T] : S^\omega \to \{0,1\}$ be the random variable that indicates that the n-th state is not in the target set T. Note that $[s_n \notin T] \geq \mathtt{val}(s_n)$ because target states have value 0. We have:

$$\begin{array}{rcll}
\mathcal{P}_{\mathcal{M},s_0,\sigma}(\mathtt{Safety}(T)) & = & \mathcal{P}_{\mathcal{M},s_0,\sigma}\left(\bigwedge_{i=0}^{\infty} \mathsf{X}^i \neg T\right) & \text{semantics of } \mathtt{Safety}(T) = \mathsf{G}\neg T \\
& = & \lim_{n\to\infty} \mathcal{P}_{\mathcal{M},s_0,\sigma}\left(\bigwedge_{i=0}^{n} \mathsf{X}^i \neg T\right) & \text{continuity of measures} \\
& = & \lim_{n\to\infty} \mathcal{P}_{\mathcal{M},s_0,\sigma}(\mathsf{X}^n \neg T) & T \text{ is a sink} \\
& = & \lim_{n\to\infty} \mathcal{E}([s_n \notin T]) & \text{definition of } [s_n \notin T] \\
& \geq & \liminf_{n\to\infty} \mathcal{E}(\mathtt{val}(s_n)) & \text{as } [s_n \notin T] \geq \mathtt{val}(s_n) \\
& \geq & \mathtt{val}(s_0) - \varepsilon & \text{Equation (10).} \quad \blacktriangleleft
\end{array}$$

We can now combine Theorem 9 with the results from [15] to show the existence of MD strategies assuming universal transience.

▶ **Theorem 10.** *For universally transient MDPs optimal strategies for* $\{0,1,2\}$-$\mathtt{Parity}$*, where they exist, can be chosen uniformly MD.*

Formally, let $\mathcal{M}$ *be a universally transient MDP with states* S*,* $Col : S \to \{0,1,2\}$*, and* $\varphi = \mathtt{Parity}(Col)$*. There exists an MD strategy* σ' *that is optimal for all states* s *that have an optimal strategy:* $\big(\exists \sigma \in \Sigma.\, \mathcal{P}_{\mathcal{M},s,\sigma}(\varphi) = \mathtt{val}_{\mathcal{M}}(s)\big) \implies \mathcal{P}_{\mathcal{M},s,\sigma'}(\varphi) = \mathtt{val}_{\mathcal{M}}(s)$*.*

Proof. Let $\mathcal{M}_+$ be the conditioned version of $\mathcal{M}$ w.r.t. φ (see [15, Def. 19] for a precise definition). By Lemma 17, $\mathcal{M}_+$ is still a universally transient MDP and therefore by Theorem 9, there exist uniformly ε-optimal MD strategies for every safety objective and every $\varepsilon > 0$. The claim now follows from [15, Theorem 22]. ◀

▶ **Theorem 11.** *For every universally transient countable MDP* $\mathcal{M}$*, co-Büchi objective and* $\varepsilon > 0$ *there exists a uniformly* ε*-optimal MD strategy.*

Formally, let $\mathcal{M}$ *be a universally transient countable MDP with states* S*,* $Col : S \to \{0,1\}$ *be a coloring,* $\varphi = \mathtt{Parity}(Col)$ *and* $\varepsilon > 0$*.*

There exists an MD strategy σ' *s.t. for every state* s*,* $\mathcal{P}_{\mathcal{M},s,\sigma'}(\varphi) \geq \mathtt{val}_{\mathcal{M}}(s) - \varepsilon$*.*

Proof. This directly follows from Theorem 9 and [15, Theorem 25]. ◀

6 The Conditioned MDP

Given an MDP $\mathcal{M}$ and an objective φ that is tail in $\mathcal{M}$, a construction of a *conditioned* MDP $\mathcal{M}_+$ was provided in [17, Lemma 6] that, very loosely speaking, "scales up" the probability of φ so that any strategy σ is optimal in $\mathcal{M}$ if it is almost surely winning in $\mathcal{M}_+$. For certain tail objectives, this construction was used in [17] to reduce the sufficiency of MD strategies for *optimal* strategies to the sufficiency of MD strategies for *almost surely winning* strategies, which is a special case that may be easier to handle.

However, the construction was restricted to states that *have* an optimal strategy. In fact, states in $\mathcal{M}$ that do not have an optimal strategy do not appear in $\mathcal{M}_+$. In the following, we lift this restriction by constructing a more general version of the conditioned MDP, called $\mathcal{M}_*$. The MDP $\mathcal{M}_*$ will contain all states from $\mathcal{M}$ that have a positive value w.r.t. φ in $\mathcal{M}$. Moreover, all these states will have value 1 in $\mathcal{M}_*$. It will then follow from Lemma 13(3) below that an ε-optimal strategy in $\mathcal{M}_*$ is $\varepsilon\mathtt{val}_{\mathcal{M}}(s_0)$-optimal in $\mathcal{M}$. This allows us to reduce the sufficiency of MD strategies for ε-optimal strategies to the sufficiency of MD strategies for ε-optimal strategies for states with value 1. In fact, it also follows that if an MD strategy σ is uniform ε-optimal in $\mathcal{M}_*$, it is *multiplicatively* uniform ε-optimal in $\mathcal{M}$, i.e., $\mathcal{P}_{\mathcal{M},s,\sigma}(\varphi) \geq (1-\varepsilon) \cdot \mathtt{val}_{\mathcal{M}}(s)$ holds for all states s.

▶ **Definition 12.** *For an MDP $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ and an objective φ that is tail in $\mathcal{M}$, define the* conditioned version *of $\mathcal{M}$ w.r.t. φ to be the MDP $\mathcal{M}_* = (S_*, S_{*\Box}, S_{*\bigcirc}, \longrightarrow_*, P_*)$ with*

$$\begin{aligned}
S_{*\Box} &= \{s \in S_\Box \mid \mathtt{val}_{\mathcal{M}}(s) > 0\} \\
S_{*\bigcirc} &= \{s \in S_\bigcirc \mid \mathtt{val}_{\mathcal{M}}(s) > 0\} \cup \{s_\perp\} \cup \{(s,t) \in \longrightarrow \mid s \in S_\Box,\ \mathtt{val}_{\mathcal{M}}(s) > 0\} \\
\longrightarrow_* &= \{(s,(s,t)) \in (S_\Box \times \longrightarrow) \mid \mathtt{val}_{\mathcal{M}}(s) > 0,\ s {\longrightarrow} t\} \cup \\
&\quad \{(s,t) \in S_\bigcirc \times S \mid \mathtt{val}_{\mathcal{M}}(s) > 0,\ \mathtt{val}_{\mathcal{M}}(t) > 0\} \cup \\
&\quad \{((s,t),t) \in (\longrightarrow \times S) \mid \mathtt{val}_{\mathcal{M}}(s) > 0,\ \mathtt{val}_{\mathcal{M}}(t) > 0\} \cup \\
&\quad \{((s,t),s_\perp) \in (\longrightarrow \times \{s_\perp\}) \mid \mathtt{val}_{\mathcal{M}}(s) > \mathtt{val}_{\mathcal{M}}(t)\} \cup \\
&\quad \{(s_\perp, s_\perp)\}
\end{aligned}$$

$$P_*(s,t) = P(s,t) \cdot \frac{\mathtt{val}_{\mathcal{M}}(t)}{\mathtt{val}_{\mathcal{M}}(s)} \qquad P_*((s,t),t) = \frac{\mathtt{val}_{\mathcal{M}}(t)}{\mathtt{val}_{\mathcal{M}}(s)}$$

$$P_*((s,t),s_\perp) = 1 - \frac{\mathtt{val}_{\mathcal{M}}(t)}{\mathtt{val}_{\mathcal{M}}(s)} \qquad P_*(s_\perp, s_\perp) = 1$$

for a fresh state $s_\perp$.

The conditioned MDP is well-defined. Indeed, as φ is tail in $\mathcal{M}$, for any $s \in S_\bigcirc$ we have $\mathtt{val}_{\mathcal{M}}(s) = \sum_{s \longrightarrow t} P(s,t)\mathtt{val}_{\mathcal{M}}(t)$, and so if $\mathtt{val}_{\mathcal{M}}(s) > 0$ then $\sum_{s \longrightarrow t} P_*(s,t) = 1$.

▶ **Lemma 13.** *Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ be an MDP, and let φ be an objective that is tail in $\mathcal{M}$. Let $\mathcal{M}_* = (S_*, S_{*\Box}, S_{*\bigcirc}, \longrightarrow_*, P_*)$ be the conditioned version of $\mathcal{M}$ w.r.t. φ. Let $s_0 \in S_* \cap S$. Let $\sigma \in \Sigma_{\mathcal{M}_*}$, and note that σ can be transformed to a strategy in $\mathcal{M}$ in a natural way. Then:*

1. *For all $n \geq 0$ and all partial runs $s_0 s_1 \cdots s_n \in s_0 S_*^*$ in $\mathcal{M}_*$ with $s_n \in S$:*

$$\mathtt{val}_{\mathcal{M}}(s_0) \cdot \mathcal{P}_{\mathcal{M}_*,s_0,\sigma}(s_0 s_1 \cdots s_n S_*^\omega) \;=\; \mathcal{P}_{\mathcal{M},s_0,\sigma}(\overline{s_0 s_1 \cdots s_n} S^\omega) \cdot \mathtt{val}_{\mathcal{M}}(s_n)\,,$$

   where $\overline{w}$ for a partial run w in $\mathcal{M}_*$ refers to its natural contraction to a partial run in $\mathcal{M}$; i.e., $\overline{w}$ is obtained from w by deleting all states of the form (s,t).

2. *For all measurable $\mathfrak{R} \subseteq s_0(S_* \setminus \{s_\perp\})^\omega$ we have*

$$\mathcal{P}_{\mathcal{M},s_0,\sigma}(\overline{\mathfrak{R}}) \;\geq\; \mathtt{val}_{\mathcal{M}}(s_0) \cdot \mathcal{P}_{\mathcal{M}_*,s_0,\sigma}(\mathfrak{R}) \;\geq\; \mathcal{P}_{\mathcal{M},s_0,\sigma}(\overline{\mathfrak{R}} \cap [\![\varphi]\!]^{s_0})\,,$$

   where $\overline{\mathfrak{R}}$ is obtained from $\mathfrak{R}$ by deleting, in all runs, all states of the form (s,t).

3. *We have $\mathtt{val}_{\mathcal{M}}(s_0) \cdot \mathcal{P}_{\mathcal{M}_*,s_0,\sigma}(\varphi) = \mathcal{P}_{\mathcal{M},s_0,\sigma}(\varphi)$. In particular, $\mathtt{val}_{\mathcal{M}_*}(s_0) = 1$, and, for any $\varepsilon \geq 0$, strategy σ is ε-optimal in $\mathcal{M}_*$ if and only if it is $\varepsilon\mathtt{val}_{\mathcal{M}}(s_0)$-optimal in $\mathcal{M}$.*

Lemma 13.3 provides a way of proving the existence of MD strategies that attain, for each state s, a fixed *fraction* (arbitrarily close to 1) of the value of s:

▶ **Theorem 14.** *Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ be an MDP, and let φ be an objective that is tail in $\mathcal{M}$. Let $\mathcal{M}_* = (S_*, S_{*\Box}, S_{*\bigcirc}, \longrightarrow_*, P_*)$ be the conditioned version of $\mathcal{M}$ w.r.t. φ. Let $\varepsilon \geq 0$. Any MD strategy σ that is uniformly ε-optimal in $\mathcal{M}_*$ (i.e., $\mathcal{P}_{\mathcal{M}_*,s,\sigma}(\varphi) \geq \mathtt{val}_{\mathcal{M}_*}(s) - \varepsilon$ holds for all $s \in S_*$) is* multiplicatively ε-optimal *in $\mathcal{M}$ (i.e., $\mathcal{P}_{\mathcal{M},s,\sigma}(\varphi) \geq (1-\varepsilon)\mathtt{val}_{\mathcal{M}}(s)$ holds for all $s \in S$).*

Proof. Immediate from Lemma 13.3. ◀

As an application of Theorem 14, we can strengthen the first statement of Theorem 8 towards *multiplicatively* (see Theorem 14) uniform ε-optimal MD strategies for `Transience`.

▶ **Corollary 15.** *In every countable MDP there exist multiplicatively uniform ε-optimal MD strategies for* `Transience`*.*

Proof. Let $\mathcal{M}$ be a countable MDP, and $\mathcal{M}_*$ its conditioned version w.r.t. `Transience`. Let $\varepsilon > 0$. By Theorem 8, there is a uniform ε-optimal MD strategy σ for `Transience` in $\mathcal{M}_*$. By Theorem 14, strategy σ is multiplicatively uniform ε-optimal in $\mathcal{M}$. ◀

The following lemma, stating that universal transience is closed under "conditioning", is needed for the proof of Lemma 17 below.

▶ **Lemma 16.** *Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P)$ be an MDP, and let φ be an objective that is tail in $\mathcal{M}$. Let $\mathcal{M}_* = (S_*, S_{*\Box}, S_{*\bigcirc}, \longrightarrow_*, P_*)$ be the conditioned version of $\mathcal{M}$ w.r.t. φ, where $s_\perp$ is replaced by an infinite chain $s_\perp^1 \longrightarrow s_\perp^2 \longrightarrow \cdots$. If $\mathcal{M}$ is universally transient, then so is $\mathcal{M}_*$.*

In [17, Lemma 6] a variant, say $\mathcal{M}_+$, of the conditioned MDP $\mathcal{M}_*$ from Definition 12 was proposed. This variant $\mathcal{M}_+$ differs from $\mathcal{M}_*$ in that $\mathcal{M}_+$ has only those states s from $\mathcal{M}$ that have an optimal strategy, i.e., a strategy σ with $\mathcal{P}_{\mathcal{M},s,\sigma}(\varphi) = \mathtt{val}_{\mathcal{M}}(s)$. Further, for any transition $s \longrightarrow t$ in $\mathcal{M}_+$ where s is a controlled state, we have $\mathtt{val}_{\mathcal{M}}(s) = \mathtt{val}_{\mathcal{M}}(t)$, i.e., $\mathcal{M}_+$ does not have value-decreasing transitions emanating from controlled states. The following lemma was used in the proof of Theorem 10:

▶ **Lemma 17.** *Let $\mathcal{M}$ be an MDP, and let φ be an objective that is tail in $\mathcal{M}$. Let $\mathcal{M}_+$ be the conditioned version w.r.t. φ in the sense of [17, Lemma 6]. If $\mathcal{M}$ is universally transient, then so is $\mathcal{M}_+$.*

7 Conclusion

The `Transience` objective admits ε-optimal (resp. optimal) MD strategies even in *infinitely* branching MDPs. This is unusual, since ε-optimal strategies for most other objectives require infinite memory if the MDP is infinitely branching (in particular all objectives generalizing Safety [17]).

`Transience` encodes a notion of continuous progress, which can be used as a tool to reason about the strategy complexity of other objectives in countable MDPs. E.g., our result on `Transience` is used in [18] as a building block to show upper bounds on the strategy complexity of certain threshold objectives w.r.t. mean payoff, total payoff and point payoff.

References

1 Pieter Abbeel and Andrew Y. Ng. Learning first-order Markov models for control. In *Advances in Neural Information Processing Systems 17*. MIT Press, 2004. URL: `http://papers.nips.cc/paper/2569-learning-first-order-markov-models-for-control`.

2 Galit Ashkenazi-Golan, János Flesch, Arkadi Predtetchinski, and Eilon Solan. Reachability and safety objectives in Markov decision processes on long but finite horizons. *Journal of Optimization Theory and Applications*, 2020.

3 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

4 Patrick Billingsley. *Probability and Measure*. Wiley, 1995. Third Edition.

5 Vincent D. Blondel and John N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 2000.

6 Nicole Bäuerle and Ulrich Rieder. *Markov Decision Processes with Applications to Finance*. Springer-Verlag Berlin Heidelberg, 2011.

7 K. Chatterjee and T. Henzinger. A survey of stochastic ω-regular games. *Journal of Computer and System Sciences*, 2012.

8 Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018. `doi:10.1007/978-3-319-10575-8`.

9 E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, December 1999.

10 William Feller. *An Introduction to Probability Theory and Its Applications*. Wiley & Sons, second edition, 1966.

11 János Flesch, Arkadi Predtetchinski, and William Sudderth. Simplifying optimal strategies in limsup and liminf stochastic games. *Discrete Applied Mathematics*, 2018.

12 T.P. Hill and V.C. Pestien. The existence of good Markov strategies for decision processes with general payoffs. *Stoch. Processes and Appl.*, 1987.

13 S. Kiefer, R. Mayr, M. Shirmohammadi, and P. Totzke. Transience in countable MDPs. In *International Conference on Concurrency Theory*, LIPIcs, 2021. Full version at `arXiv:2012.13739`.

14 Stefan Kiefer, Richard Mayr, Mahsa Shirmohammadi, and Patrick Totzke. Büchi objectives in countable MDPs. In *International Colloquium on Automata, Languages and Programming*, LIPIcs, 2019. Full version at `arXiv:1904.11573`. `doi:10.4230/LIPIcs.ICALP.2019.119`.

15 Stefan Kiefer, Richard Mayr, Mahsa Shirmohammadi, and Patrick Totzke. Strategy Complexity of Parity Objectives in Countable MDPs. In *International Conference on Concurrency Theory*, 2020. `doi:10.4230/LIPIcs.CONCUR.2020.7`.

16 Stefan Kiefer, Richard Mayr, Mahsa Shirmohammadi, Patrick Totzke, and Dominik Wojtczak. How to play in infinite MDPs (invited talk). In *International Colloquium on Automata, Languages and Programming*, 2020. `doi:10.4230/LIPIcs.ICALP.2020.3`.

17 Stefan Kiefer, Richard Mayr, Mahsa Shirmohammadi, and Dominik Wojtczak. Parity Objectives in Countable MDPs. In *Annual IEEE Symposium on Logic in Computer Science*, 2017. `doi:10.1109/LICS.2017.8005100`.

18 Richard Mayr and Eric Munday. Strategy Complexity of Mean Payoff, Total Payoff and Point Payoff Objectives in Countable MDPs. In *International Conference on Concurrency Theory*, LIPIcs, 2021. The full version is available at `arXiv:2107.03287`.

19 A. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Computation Theory*, LNCS, 1984.

20 Donald Ornstein. On the existence of stationary optimal strategies. *Proceedings of the American Mathematical Society*, 1969. `doi:10.2307/2035700`.

21 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1st edition, 1994.

22 George Santayana. Reason in common sense, 1905. In *Volume 1 of The Life of Reason*. URL: `https://en.wikipedia.org/wiki/George_Santayana`.

23 Manfred Schäl. Markov decision processes in finance and dynamic options. In *Handbook of Markov Decision Processes*. Springer, 2002.

24 Olivier Sigaud and Olivier Buffet. *Markov Decision Processes in Artificial Intelligence*. John Wiley & Sons, 2013.

25 William D. Sudderth. Optimal Markov strategies. *Decisions in Economics and Finance*, 2020.

26 R.S. Sutton and A.G Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, 2018.

27 Moshe Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, 1985. `doi:10.1109/SFCS.1985.12`.

Strategy Complexity of Mean Payoff, Total Payoff and Point Payoff Objectives in Countable MDPs

Richard Mayr
University of Edinburgh, UK

Eric Munday
University of Edinburgh, UK

Abstract

We study countably infinite Markov decision processes (MDPs) with real-valued transition rewards. Every infinite run induces the following sequences of payoffs: 1. Point payoff (the sequence of directly seen transition rewards), 2. Total payoff (the sequence of the sums of all rewards so far), and 3. Mean payoff. For each payoff type, the objective is to maximize the probability that the $\liminf$ is non-negative. We establish the complete picture of the strategy complexity of these objectives, i.e., how much memory is necessary and sufficient for ε-optimal (resp. optimal) strategies. Some cases can be won with memoryless deterministic strategies, while others require a step counter, a reward counter, or both.

2012 ACM Subject Classification Theory of computation → Random walks and Markov chains; Mathematics of computing → Probability and statistics

Keywords and phrases Markov decision processes, Strategy complexity, Mean payoff

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.12

Related Version *Full Version*: `https://arxiv.org/abs/2107.03287`

Acknowledgements We thank an anonymous reviewer for very detailed and helpful comments.

1 Introduction

Background. Markov decision processes (MDPs) are a standard model for dynamic systems that exhibit both stochastic and controlled behavior [18]. Applications include control theory [5, 1], operations research and finance [2, 6, 20], artificial intelligence and machine learning [23, 21], and formal verification [9, 3].

An MDP is a directed graph where states are either random or controlled. In a random state the next state is chosen according to a fixed probability distribution. In a controlled state the controller can choose a distribution over all possible successor states. By fixing a strategy for the controller (and an initial state), one obtains a probability space of runs of the MDP. The goal of the controller is to optimize the expected value of some objective function on the runs. The type of strategy necessary to achieve an ε-optimal (resp. optimal) value for a given objective is called its *strategy complexity*.

Transition rewards and liminf objectives. MDPs are given a reward structure by assigning a real-valued (resp. integer or rational) reward to each transition. Every run then induces an infinite sequence of seen transition rewards $r_0 r_1 r_2 \ldots$. We consider the $\liminf$ of this sequence, as well as two other important derived sequences.

1. The point payoff considers the $\liminf$ of the sequence $r_0 r_1 r_2 \ldots$ directly.
2. The total payoff considers the $\liminf$ of the sequence $\left\{\sum_{i=0}^{n-1} r_i\right\}_{n\in\mathbb{N}}$, i.e., the sum of all rewards seen so far.
3. The mean payoff considers the $\liminf$ of the sequence $\left\{\frac{1}{n}\sum_{i=0}^{n-1} r_i\right\}_{n\in\mathbb{N}}$, i.e., the mean of all rewards seen so far in an expanding prefix of the run.

For each of the three cases above, the $\liminf$ threshold objective is to maximize the probability that the $\liminf$ of the respective type of sequence is ≥ 0.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 12; pp. 12:1–12:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our contribution. We establish the strategy complexity of all the $\liminf$ threshold objectives above for *countably infinite* MDPs. (For the simpler case of finite MDPs, see the paragraph on related work below.) We show the amount and type of memory that is sufficient for ε-optimal strategies (and optimal strategies, where they exist), and corresponding lower bounds in the sense of Remark 1. This is not only the distinction between memoryless, finite memory and infinite memory, but the type of infinite memory that is necessary and sufficient. A step counter is an integer counter that merely counts the number of steps in the run (i.e., like a discrete clock), while a reward counter is a variable that records the sum of all rewards seen so far. (The reward counter has the same type as the transition rewards in the MDP, i.e., integers, rationals or reals.) While these use infinite memory, it is a very restricted form, since this memory is not directly controlled by the player. Strategies using only a step counter are also called Markov strategies [18].

Some of the $\liminf$ objectives can be attained by memoryless deterministic (MD) strategies, while others require (in the sense of Remark 1) a step counter, a reward counter, or both. It depends on the type of objective (point, total, or mean payoff) and on whether the MDP is finitely or infinitely branching. For clarity of presentation, our counterexamples use large transition rewards and high degrees of branching. However, the lower bounds hold even for just binary branching MDPs with transition rewards in $\{-1, 0, 1\}$; cf. [17].

For our objectives, the strategy complexities of ε-optimal and optimal strategies (where they exist) coincide, but the proofs are different. Table 1 shows the results for all combinations.

Table 1 Strategy complexity of ε-optimal/optimal strategies for point, total and mean payoff objectives in infinitely/finitely branching MDPs. MD stands for memoryless deterministic, SC for step counter, RC for reward counter and SC+RC for both. All strategies are deterministic and randomization does not help. For each result, we list the numbers of the theorems that show the upper and lower bounds on the strategy complexity. The lower bounds hold in the sense of Remark 1, but work for integer rewards. The upper bounds hold even for real-valued rewards.

	Point payoff	Total payoff	Mean payoff
ε-optimal, infinitely branching	SC 17, 32	SC+RC 17, 9, 34	SC+RC 15, 8, 33
optimal, infinitely branching	SC 17, 35	SC+RC 14, 17, 35	SC+RC 13, 16, 35
ε-optimal, finitely branching	MD 27	RC 9, 30	SC+RC 15, 8, 33
optimal, finitely branching	MD 31	RC 14, 31	SC+RC 13, 16, 35

Some complex new proof techniques are developed to show these results. E.g., the examples showing the lower bound in cases where both a step counter and a reward counter are required use a finely tuned tradeoff between different risks that can be managed with both counters, but not with just one counter plus arbitrary finite memory. The strategies showing the upper bounds need to take into account convergence effects, e.g., the sequence of point rewards $-1/2, -1/3, -1/4, \ldots$ *does* satisfy $\liminf \geq 0$, i.e., one cannot assume that rewards are integers.

Due to space constraints, we sketch some proofs in the main body. Full proofs can be found in [17].

Related work. Mean payoff objectives for *finite* MDPs have been widely studied; cf. survey in [8]. There exist optimal MD strategies for $\liminf$ mean payoff (which are also optimal for $\limsup$ mean payoff since the transition rewards are bounded), and the associated computational problems can be solved in polynomial time [8, 18]. Similarly, see [7] for a survey on $\limsup$ and $\liminf$ point payoff objectives in finite stochastic games and MDPs, where there also exist optimal MD strategies, and the more recent paper by Flesch, Predtetchinski and Sudderth [11] on simplifying optimal strategies.

All this does *not* carry over to countably infinite MDPs. Optimal strategies need not exist (not even for much simpler objectives), (ε-)optimal strategies can require infinite memory, and computational problems are not defined in general, since a countable MDP need not be finitely presented [16]. Moreover, attainment for $\liminf$ mean payoff need not coincide with attainment for $\limsup$ mean payoff, even for very simple examples. E.g., consider the acyclic infinite graph with transitions $s_n \to s_{n+1}$ for all $n \in \mathbb{N}$ with reward $(-1)^n 2^n$ in the n-th step, which yields a $\liminf$ mean payoff of $-\infty$ and a $\limsup$ mean payoff of $+\infty$.

Mean payoff objectives for countably infinite MDPs have been considered in [18, Section 8.10], e.g., [18, Example 8.10.2] shows that there are no optimal MD (memoryless deterministic) strategies for $\liminf/\limsup$ mean payoff. [19, Counterexample 1.3] shows that there are not even ε-optimal memoryless randomized strategies for $\liminf/\limsup$ mean payoff. (We show much stronger lower/upper bounds; cf. Table 1.)

Sudderth [22] considered an objective on countable MDPs that is related to our point payoff threshold objective. However, instead of maximizing the probability that the $\liminf/\limsup$ is non-negative, it asks to maximize the *expectation* of the $\liminf/\limsup$ point payoffs, which is a different problem (e.g., it can tolerate a high probability of a negative $\liminf/\limsup$ if the remaining cases have a huge positive $\liminf/\limsup$). Hill & Pestien [12] showed the existence of good randomized Markov strategies for the $\limsup$ of the *expected* average reward up-to step n for growing n, and for the *expected* $\liminf$ of the point payoffs.

2 Preliminaries

Markov decision processes. A *probability distribution* over a countable set S is a function $f : S \to [0,1]$ with $\sum_{s \in S} f(s) = 1$. We write $\mathcal{D}(S)$ for the set of all probability distributions over S. A *Markov decision process* (MDP) $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P, r)$ consists of a countable set S of *states*, which is partitioned into a set $S_\Box$ of *controlled states* and a set $S_\bigcirc$ of *random states*, a *transition relation* $\longrightarrow \subseteq S \times S$, and a *probability function* $P : S_\bigcirc \to \mathcal{D}(S)$. We write $s{\longrightarrow}s'$ if $(s, s') \in \longrightarrow$, and refer to s' as a *successor* of s. We assume that every state has at least one successor. The probability function P assigns to each random state $s \in S_\bigcirc$ a probability distribution $P(s)$ over its (non-empty) set of successor states. A *sink in* $\mathcal{M}$ is a subset $T \subseteq S$ closed under the $\longrightarrow$ relation, that is, $s \in T$ and $s{\longrightarrow}s'$ implies that $s' \in T$.

An MDP is *acyclic* if the underlying directed graph $(S, \longrightarrow)$ is acyclic, i.e., there is no directed cycle. It is *finitely branching* if every state has finitely many successors and *infinitely branching* otherwise. An MDP without controlled states ($S_\Box = \emptyset$) is called a *Markov chain*.

In order to specify our mean/total/point payoff objectives (see below), we define a function $r : S \times S \to \mathbb{R}$ that assigns numeric rewards to transitions.

Strategies and Probability Measures. A *run* ρ is an infinite sequence of states and transitions $s_0 e_0 s_1 e_1 \cdots$ such that $e_i = (s_i, s_{i+1}) \in \longrightarrow$ for all $i \in \mathbb{N}$. Let $\mathit{Runs}_{\mathcal{M}}^{s_0}$ be the set of all runs from s_0 in the MDP $\mathcal{M}$. A *partial run* is a finite prefix of a run, $\mathit{pRuns}_{\mathcal{M}}^{s_0}$ is the set of all partial runs from s_0 and $\mathit{pRuns}_{\mathcal{M}}$ the set of partial runs from any state.

We write $\rho_s(i) \overset{\text{def}}{=} s_i$ for the i-th state along ρ and $\rho_e(i) \overset{\text{def}}{=} e_i$ for the i-th transition along ρ. We sometimes write runs as $s_0 s_1 \cdots$, leaving the transitions implicit. We say that a (partial) run ρ *visits* s if $s = \rho_s(i)$ for some i, and that ρ starts in s if $s = \rho_s(0)$.

A *strategy* is a function $\sigma : \mathit{pRuns}_{\mathcal{M}} \cdot S_\Box \to \mathcal{D}(S)$ that assigns to partial runs ρs, where $s \in S_\Box$, a distribution over the successors $\{s' \in S \mid s{\longrightarrow}s'\}$. The set of all strategies in $\mathcal{M}$ is denoted by $\Sigma_{\mathcal{M}}$ (we omit the subscript and write Σ if $\mathcal{M}$ is clear from the context). A (partial) run $s_0 e_0 s_1 e_1 \cdots$ is consistent with a strategy σ if for all i either $s_i \in S_\Box$ and $\sigma(s_0 e_0 s_1 e_1 \cdots s_i)(s_{i+1}) > 0$, or $s_i \in S_\bigcirc$ and $P(s_i)(s_{i+1}) > 0$.

An MDP $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P, r)$, an initial state $s_0 \in S$, and a strategy σ induce a probability space in which the outcomes are runs starting in s_0 and with measure $\mathcal{P}_{\mathcal{M},s_0,\sigma}$ defined as follows. It is first defined on *cylinders* $s_0e_0s_1e_1 \ldots s_n \mathit{Runs}^{s_n}_{\mathcal{M}}$: if $s_0e_0s_1e_1 \ldots s_n$ is not a partial run consistent with σ then $\mathcal{P}_{\mathcal{M},s_0,\sigma}(s_0e_0s_1e_1 \ldots s_n \mathit{Runs}^{s_n}_{\mathcal{M}}) \overset{\text{def}}{=} 0$. Otherwise, $\mathcal{P}_{\mathcal{M},s_0,\sigma}(s_0e_0s_1e_1 \ldots s_n \mathit{Runs}^{s_n}_{\mathcal{M}}) \overset{\text{def}}{=} \prod_{i=0}^{n-1} \bar{\sigma}(s_0e_0s_1 \ldots s_i)(s_{i+1})$, where $\bar{\sigma}$ is the map that extends σ by $\bar{\sigma}(ws) = P(s)$ for all partial runs $ws \in \mathit{pRuns}_{\mathcal{M}} \cdot S_\bigcirc$. By Carathéodory's theorem [4], this extends uniquely to a probability measure $\mathcal{P}_{\mathcal{M},s_0,\sigma}$ on the Borel σ-algebra $\mathcal{F}$ of subsets of $\mathit{Runs}^{s_0}_{\mathcal{M}}$. Elements of $\mathcal{F}$, i.e., measurable sets of runs, are called *events* or *objectives* here. For $X \in \mathcal{F}$ we will write $\overline{X} \overset{\text{def}}{=} \mathit{Runs}^{s_0}_{\mathcal{M}} \setminus X \in \mathcal{F}$ for its complement and $\mathcal{E}_{\mathcal{M},s_0,\sigma}$ for the expectation wrt. $\mathcal{P}_{\mathcal{M},s_0,\sigma}$. We drop the indices if possible without ambiguity.

Objectives. We consider objectives that are determined by a predicate on infinite runs. We assume familiarity with the syntax and semantics of the temporal logic LTL [10]. Formulas are interpreted on the structure $(S, \longrightarrow)$. We use $[\![\varphi]\!]^s$ to denote the set of runs starting from s that satisfy the LTL formula φ, which is a measurable set [24]. We also write $[\![\varphi]\!]$ for $\bigcup_{s \in S} [\![\varphi]\!]^s$. Where it does not cause confusion we will identify φ and $[\![\varphi]\!]$ and just write $\mathcal{P}_{\mathcal{M},s,\sigma}(\varphi)$ instead of $\mathcal{P}_{\mathcal{M},s,\sigma}([\![\varphi]\!]^s)$. The reachability objective of eventually visiting a set of states X can be expressed by $[\![\mathsf{F}X]\!] \overset{\text{def}}{=} \{\rho \mid \exists i.\, \rho_s(i) \in X\}$. Reaching X within at most k steps is expressed by $[\![\mathsf{F}^{\leq k}X]\!] \overset{\text{def}}{=} \{\rho \mid \exists i \leq k.\, \rho_s(i) \in X\}$. The definitions for eventually visiting certain transitions are analogous. The operator G (always) is defined as $\neg\mathsf{F}\neg$. So the safety objective of avoiding X is expressed by $\mathsf{G}\neg X$.

- The $\mathit{PP}_{\liminf \geq 0}$ objective is to maximize the probability that the lim inf of the *point* payoffs (the immediate transition rewards) is ≥ 0, i.e., $\mathit{PP}_{\liminf \geq 0} \overset{\text{def}}{=} \{\rho \mid \liminf_{n \in \mathbb{N}} r(\rho_e(n)) \geq 0\}$.
- The $\mathit{TP}_{\liminf \geq 0}$ objective is to maximize the probability that the lim inf of the *total* payoff (the sum of the transition rewards seen so far) is ≥ 0, i.e., $\mathit{TP}_{\liminf \geq 0} \overset{\text{def}}{=} \{\rho \mid \liminf_{n \in \mathbb{N}} \sum_{j=0}^{n-1} r(\rho_e(j)) \geq 0\}$.
- The $\mathit{MP}_{\liminf \geq 0}$ objective is to maximize the probability that the lim inf of the *mean* payoff is ≥ 0, i.e., $\mathit{MP}_{\liminf \geq 0} \overset{\text{def}}{=} \{\rho \mid \liminf_{n \in \mathbb{N}} \frac{1}{n} \sum_{j=0}^{n-1} r(\rho_e(j)) \geq 0\}$.

An objective φ is called *tail* in $\mathcal{M}$ if for every run $\rho'\rho$ in $\mathcal{M}$ with some finite prefix ρ' we have $\rho'\rho \in [\![\varphi]\!] \Leftrightarrow \rho \in [\![\varphi]\!]$. An objective is called a *tail objective* if it is tail in every MDP. $\mathit{PP}_{\liminf \geq 0}$ and $\mathit{MP}_{\liminf \geq 0}$ are tail objectives, but $\mathit{TP}_{\liminf \geq 0}$ is not. Also $\mathit{PP}_{\liminf \geq 0}$ is more general than co-Büchi. (The special case of integer transition rewards coincides with co-Büchi, since rewards ≤ -1 and accepting states can be encoded into each other.)

Strategy Classes. Strategies are in general *randomized* (R) in the sense that they take values in $\mathcal{D}(S)$. A strategy σ is *deterministic* (D) if $\sigma(\rho)$ is a Dirac distribution for all ρ. General strategies can be *history dependent* (H), while others are restricted by the size or type of memory they use, see below. We consider certain classes of strategies:

- A strategy σ is *memoryless* (M) (also called *positional*) if it can be implemented with a memory of size 1. We may view M-strategies as functions $\sigma : S_\Box \to \mathcal{D}(S)$.
- A strategy σ is *finite memory* (F) if there exists a finite memory M implementing σ. Hence FR stands for finite memory randomized.
- A *step counter strategy* bases decisions only on the current state and the number of steps taken so far, i.e., it uses an unbounded integer counter that gets incremented by 1 in every step. Such strategies are also called *Markov strategies* [18].
- *k-bit Markov strategies* use k extra bits of general purpose memory in addition to a step counter [15].

- A *reward counter strategy* uses infinite memory, but only in the form of a counter that always contains the sum of all transition rewards seen to far.
- A *step counter + reward counter strategy* uses both a step counter and a reward counter.

See [17] for a formal definition how strategies use memory. Step counters and reward counters are very restricted forms of memory, since the memory update is not directly under the control of the player. These counters merely record an aspect of the partial run.

Optimal and ε-optimal Strategies. Given an objective φ, the value of state s in an MDP $\mathcal{M}$, denoted by $\mathtt{val}_{\mathcal{M},\varphi}(s)$, is the supremum probability of achieving φ. Formally, $\mathtt{val}_{\mathcal{M},\varphi}(s) \overset{\text{def}}{=} \sup_{\sigma\in\Sigma} \mathcal{P}_{\mathcal{M},s,\sigma}(\varphi)$ where Σ is the set of all strategies. For $\varepsilon \geq 0$ and state $s \in S$, we say that a strategy is *ε-optimal* from s if $\mathcal{P}_{\mathcal{M},s,\sigma}(\varphi) \geq \mathtt{val}_{\mathcal{M},\varphi}(s) - \varepsilon$. A 0-optimal strategy is called *optimal*. An optimal strategy is *almost-surely winning* if $\mathtt{val}_{\mathcal{M},\varphi}(s) = 1$. Considering an MD strategy as a function $\sigma : S_\Box \to S$ and $\varepsilon \geq 0$, σ is *uniformly* ε-optimal (resp. uniformly optimal) if it is ε-optimal (resp. optimal) from *every* $s \in S$.

▶ Remark 1. To establish an upper bound X on the strategy complexity of an objective φ in countable MDPs, it suffices to prove that there always exist good (ε-optimal, resp. optimal) strategies in class X (e.g., MD, MR, FD, FR, etc.) for objective φ.

Lower bounds on the strategy complexity of an objective φ can only be established in the sense of proving that good strategies for φ do not exist in some classes Y, Z, etc. Classes of strategies that use different types of *restricted* infinite memory are generally not comparable, e.g., step counter strategies are incomparable to reward counter strategies. In particular, there is no weakest type of infinite memory with restricted use. Therefore statements like "good strategies for objective φ require at least a step counter" are always *relative* to the considered alternative strategy classes. In this paper, we only consider the strategy classes of memoryless, finite memory, step counter, reward counter and *combinations thereof*. Thus, when we write in Table 1 that an objective requires a step counter (SC), it just means that a reward counter (RC) plus finite memory is not sufficient.

For our upper bounds, we use deterministic strategies. Moreover, we show that allowing randomization does not help to reduce the strategy complexity, in the sense of Remark 1.

3 When is a step counter not sufficient?

In this section we will prove that strategies with a step counter plus arbitrary finite memory are not sufficient for ε-optimal strategies for $MP_{\liminf\geq 0}$ or $TP_{\liminf\geq 0}$. We will construct an acyclic MDP where the step counter is implicit in the state such that ε-optimal strategies for $MP_{\liminf\geq 0}$ and $TP_{\liminf\geq 0}$ still require infinite memory.

3.1 Epsilon-optimal strategies

We construct an acyclic MDP $\mathcal{M}$ in which the step counter is implicit in the state as follows.

The system consists of a sequence of gadgets. Figure 1 depicts a typical building block in this system. The system consists of these gadgets chained together as illustrated in Figure 2, starting with n sufficiently high at $n = N^*$. In the controlled choice, there is a small chance in all but the top choice of falling into a $\perp$ state. These $\perp$ states are abbreviations for an infinite chain of states with -1 reward on the transitions and are thus losing. The intuition behind the construction is that there is a random transition with branching degree $k(n) + 1$. Then, the only way to win, in the controlled states, is to play the i-th choice if one arrived from the i-th choice. Thus intuitively, to remember what this choice was, one requires at least $k(n) + 1$ memory modes. That is to say, the one and only way to win is to mimic, and mimicry requires memory.

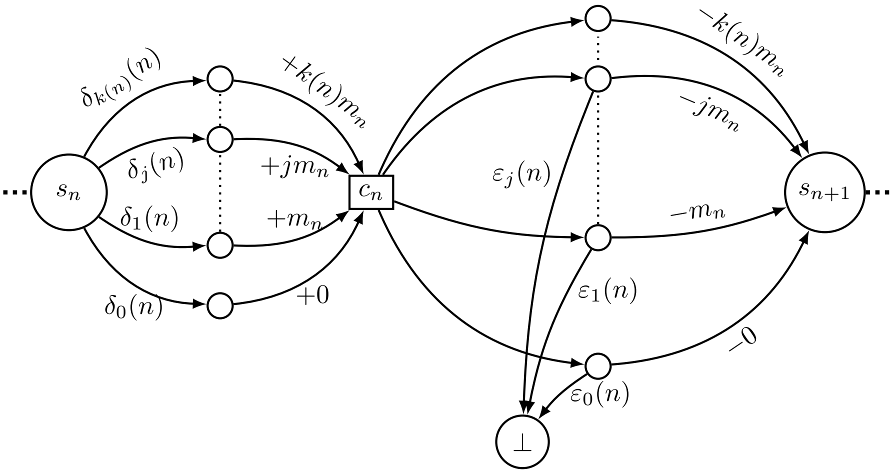


Figure 1 A typical building block with $k(n)+1$ choices, first random then controlled. The number of choices $k(n)+1$ grows unboundedly with n. This is the n-th building block of the MDP in Figure 2. The $\delta_i(n)$ and $\varepsilon_i(n)$ are probabilities depending on n and the $\pm im_n$ are transition rewards. We index the successor states of s_n and c_n from 0 to $k(n)$ to match the indexing of the δ's and ε's such that the bottom state is indexed with 0 and the top state with $k(n)$.

▶ Remark 2. $\mathcal{M}$ is acyclic, finitely branching and for every state $s \in S, \exists n_s \in \mathbb{N}$ such that every path from s_0 to s has length n_s. That is to say the step counter is implicit in the state.

Additionally, the number of transitions in each gadget now grows unboundedly with n according to the function $k(n)$. Consequently, we will show that the number of memory modes required to play correctly grows above every finite bound. This will imply that no finite amount of memory suffices for ε-optimal strategies.

Notation. All logarithms are assumed to be in base e.

$$\log_1 n \overset{\text{def}}{=} \log n, \quad \log_{i+1} n \overset{\text{def}}{=} \log(\log_i n)$$

$$\delta_0(n) \overset{\text{def}}{=} \frac{1}{\log n}, \quad \delta_i(n) \overset{\text{def}}{=} \frac{1}{\log_{i+1} n}, \quad \delta_{k(n)}(n) \overset{\text{def}}{=} 1 - \sum_{j=0}^{k(n)-1} \delta_j(n)$$

$$\varepsilon_0(n) \overset{\text{def}}{=} \frac{1}{n \log n}, \ \varepsilon_{i+1}(n) \overset{\text{def}}{=} \frac{\varepsilon_i(n)}{\log_{i+2} n}, \ \text{i.e. } \varepsilon_i(n) = \frac{1}{n \cdot \log n \cdot \log_2 n \cdots \log_{i+1} n}, \varepsilon_{k(n)}(n) \overset{\text{def}}{=} 0$$

$$\text{Tower}(0) \overset{\text{def}}{=} e^0 = 1, \quad \text{Tower}(i+1) \overset{\text{def}}{=} e^{\text{Tower}(i)}, \quad N_i \overset{\text{def}}{=} \text{Tower}(i)$$

▶ **Lemma 3.** *The family of series $\sum_{n>N_j} \delta_j(n) \cdot \varepsilon_i(n)$ is divergent for all $i, j \in \mathbb{N}$, $i < j$. Additionally, the related family of series $\sum_{n>N_i} \delta_i(n) \cdot \varepsilon_i(n)$ is convergent for all $i \in \mathbb{N}$.*

Proof. These are direct consequences of Cauchy's Condensation Test. ◀

▶ **Definition 4.** *We define $k(n)$, the rate at which the number of transitions grows. We define $k(n)$ in terms of fast growing functions g, Tower and h defined for $i \geq 1$ as follows:*

$$g(i) \overset{def}{=} min\left\{N : \left(\sum_{n>N} \delta_{i-1}(n)\varepsilon_{i-1}(n)\right) \leq 2^{-i}\right\}, \quad h(1) \overset{def}{=} 2$$

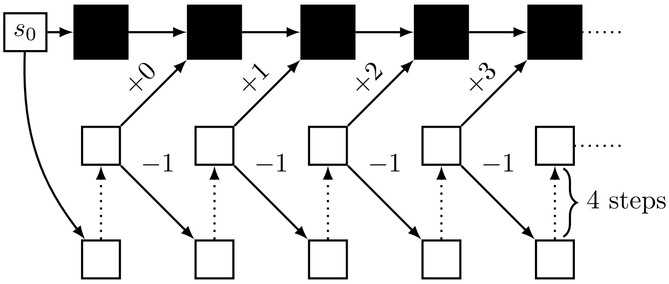


Figure 2 The buildings blocks from Figure 1 represented by black boxes are chained together (n increases as you go to the right). The chain of white boxes allows to skip arbitrarily long prefixes while preserving path length. The positive rewards from the white states to the black boxes reimburse the lost reward accumulated until then. The -1 rewards between white states ensure that skipping gadgets forever is losing.

$$h(i+1) \overset{def}{=} \left\lceil max\left\{g(i+1), Tower(i+2), min\left\{m+1 \in \mathbb{N} : \sum_{n=h(i)}^{m} \varepsilon_{i-1}(n) \geq 1\right\}\right\}\right\rceil .$$

Note that function g is well defined by Lemma 3, and $h(i+1)$ is well defined since for all i, $\sum_{n=h(i)}^{\infty} \varepsilon_{i-1}(n)$ diverges to infinity. $k(n)$ is a slow growing unbounded step function defined in terms of h as $k(n) \overset{def}{=} h^{-1}(n)$. The Tower function features in the definition to ensure that the transition probabilities are always well defined. g and h are used to smooth the proofs of e.g. Lemma 6. Notation: $N^* \overset{def}{=} min\{n \in \mathbb{N} : k(n) = 1\}$. *This is intuitively the first natural number for which the construction is well defined.*

The reward m_n which appears in the n-th gadget is defined such that it outweighs any possible reward accumulated up to that point in previous gadgets. As such we define $m_n \overset{def}{=} 2k(n)\sum_{i=N^*}^{n-1} m_i$, with $m_{N^*} \overset{def}{=} 1$ and where $k(n)$ is the branching degree.

To simplify the notation, the state s_0 in our theorem statements refers to s_{N^*}.

▶ **Lemma 5.** *For $k(n) \geq 1$, the transition probabilities in the gadgets are well defined.*

▶ **Lemma 6.** *For every $\varepsilon > 0$, there exists a strategy σ_ε with $\mathcal{P}_{\mathcal{M},s_0,\sigma_\varepsilon}(MP_{\liminf\geq 0}) \geq 1-\varepsilon$ that cannot fail unless it hits a $\bot$ state. Formally, $\mathcal{P}_{\mathcal{M},s_0,\sigma_\varepsilon}(MP_{\liminf\geq 0} \wedge \mathsf{G}(\neg \bot)) = \mathcal{P}_{\mathcal{M},s_0,\sigma_\varepsilon}(\mathsf{G}(\neg \bot)) \geq 1-\varepsilon$. So in particular, $\mathtt{val}_{\mathcal{M},MP_{\liminf\geq 0}}(s_0) = 1$.*

Proof sketch. We define a strategy σ which in c_n always mimics the choice in s_n. Playing according to σ, the only way to lose is by dropping into the $\bot$ state. This is because by mimicking, the player finishes each gadget with a reward of 0. From s_0, the probability of surviving while playing in all the gadgets is

$$\prod_{n \geq N^*} \left(1 - \sum_{j=0}^{k(n)-1} \delta_j(n) \cdot \varepsilon_j(n)\right) > 0.$$

Hence the player has a non zero chance of winning when playing σ.

When playing with the ability to skip gadgets, as illustrated in Figure 2, all runs not visiting a $\bot$ state are winning since the total reward never dips below 0. We then consider the strategy σ_ε which plays like σ after skipping forwards by sufficiently many gadgets (starting at $n \gg N^*$). Its probability of satisfying $MP_{\liminf\geq 0}$ corresponds to a tail of the above product, which can be made arbitrarily close to 1 (and thus $\geq 1-\varepsilon$). Thus the strategies σ_ε for arbitrarily small $\varepsilon > 0$ witness that $\mathtt{val}_{\mathcal{M},MP_{\liminf\geq 0}}(s_0) = 1$. ◀

▶ **Lemma 7.** *For any FR strategy σ, almost surely either the mean payoff dips below -1 infinitely often, or the run hits a $\bot$ state, i.e. $\mathcal{P}_{\mathcal{M},\sigma,s_0}(MP_{\liminf \geq 0}) = 0$.*

Proof sketch. Let σ be some FR strategy with k memory modes. We prove a *lower bound* e_n on the probability of a local error (reaching a $\bot$ state, or seeing a mean payoff ≤ -1) in the current n-th gadget. This lower bound e_n holds regardless of events in past gadgets, regardless of the memory mode of σ upon entering the n-th gadget, and cannot be improved by σ randomizing its memory updates.

The main idea is that, once $k(n) > k + 1$ (which holds for $n \geq N'$ sufficiently large) by the Pigeonhole Principle there will always be a memory mode confusing at least two different branches $i(n), j(n) \neq k(n)$ of the previous random choice at state s_n. This confusion yields a probability $\geq e_n$ of reaching a $\bot$ state or seeing a mean payoff ≤ -1, regardless of events in past gadgets and regardless of the memory upon entering the n-th gadget. We show that $\sum_{n \geq N'} e_n$ is a *divergent* series. Thus, $\prod_{n \geq N'}(1 - e_n) = 0$. Hence, $\mathcal{P}_{\mathcal{M},\sigma,s_0}(MP_{\liminf \geq 0}) \leq \prod_{n \geq N'}(1 - e_n) = 0$. ◀

Lemma 6 and Lemma 7 yield the following theorem.

▶ **Theorem 8.** *There exists a countable, finitely branching and acyclic MDP $\mathcal{M}$ whose step counter is implicit in the state for which $\texttt{val}_{\mathcal{M},MP_{\liminf \geq 0}}(s_0) = 1$ and any FR strategy σ is such that $\mathcal{P}_{\mathcal{M},s_0,\sigma}(MP_{\liminf \geq 0}) = 0$. In particular, there are no ε-optimal k-bit Markov strategies for any $k \in \mathbb{N}$ and any $\varepsilon < 1$ for $MP_{\liminf \geq 0}$ in countable MDPs.*

All of the above results/proofs also hold for $TP_{\liminf \geq 0}$, giving us the following theorem.

▶ **Theorem 9.** *There exists a countable, finitely branching and acyclic MDP $\mathcal{M}$ whose step counter is implicit in the state for which $\texttt{val}_{\mathcal{M},TP_{\liminf \geq 0}}(s_0) = 1$ and any FR strategy σ is such that $\mathcal{P}_{\mathcal{M},s_0,\sigma}(TP_{\liminf \geq 0}) = 0$. In particular, there are no ε-optimal k-bit Markov strategies for any $k \in \mathbb{N}$ and any $\varepsilon < 1$ for $TP_{\liminf \geq 0}$ in countable MDPs.*

3.2 Optimal strategies

Even for acyclic MDPs with the step counter implicit in the state, optimal (and even almost sure winning) strategies for $MP_{\liminf \geq 0}$ require infinite memory. To prove this, we consider a variant of the MDP from the previous section which has been augmented to include restarts from the $\bot$ states. For the rest of the section, $\mathcal{M}$ is the MDP constructed in Figure 3.

▶ Remark 10. $\mathcal{M}$ is acyclic, finitely branching and the step counter is implicit in the state. We now refer to the rows of Figure 3 as gadgets, i.e., a gadget is a single instance of Figure 2 where the $\bot$ states lead to the next row.

▶ **Lemma 11.** *There exists a strategy σ such that $\mathcal{P}_{\mathcal{M},\sigma,s_0}(MP_{\liminf \geq 0}) = 1$.*

Proof sketch. Recall the strategy $\sigma_{1/2}$ defined in Lemma 6 which achieves at least $1/2$ in each gadget that it is played in. We then construct the almost surely winning strategy σ by concatenating $\sigma_{1/2}$ strategies in the sense that σ plays just like $\sigma_{1/2}$ in each gadget from each gadget's start state.

Since σ achieves at least $1/2$ in every gadget that it sees, with probability 1, runs generated by σ restart only finitely many times. The intuition is then that a run restarting finitely many times must spend an infinite tail in some final gadget. Since σ mimics in every controlled state, not restarting anymore directly implies that the total payoff is eventually always ≥ 0. Hence all runs generated by σ and restarting only finitely many times satisfy $MP_{\liminf \geq 0}$. Therefore all but a nullset of runs generated by σ are winning, i.e. $\mathcal{P}_{\mathcal{M},s_0,\sigma}(MP_{\liminf \geq 0}) = 1$. ◀

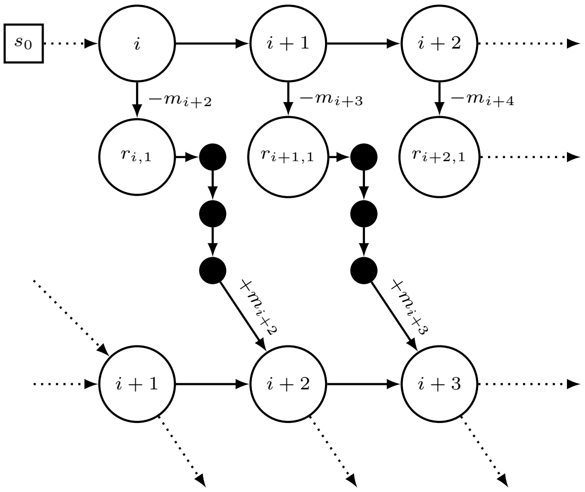


Figure 3 Each row represents a copy of the MDP depicted in Figure 2. Each white circle labeled with a number i represents the correspondingly numbered gadget (like in Figure 1) from that MDP. Now, instead of the bottom states in each gadget leading to an infinite losing chain, they lead to a restart state $r_{i,j}$ which leads to a fresh copy of the MDP (in the next row). Each restart incurs a penalty guaranteeing that the mean payoff dips below -1 before refunding it and continuing on in the next copy of the MDP. The states $r_{i,j}$ are labeled such that the j indicates that if a run sees this state, then it is the jth restart. The i indicates that the run entered the restart state from the ith gadget of the current copy of the MDP. The black states are dummy states inserted in order to preserve path length throughout.

▶ **Lemma 12.** *For any FR strategy σ, $\mathcal{P}_{\mathcal{M},\sigma,s_0}(MP_{\liminf\geq 0}) = 0$.*

Proof sketch. Let σ be any FR strategy. We partition the runs generated by σ into runs restarting infinitely often, and those restarting only finitely many times. Any runs restarting infinitely often are losing by construction. Those runs restarting only finitely many times, once in the gadget they spend an infinite tail in, let the mean payoff dip below -1 infinitely many times with probability 1 by Lemma 7. Hence we have that $\mathcal{P}_{\mathcal{M},\sigma,s_0}(MP_{\liminf\geq 0}) = 0$. ◀

From Lemma 11 and Lemma 12 we obtain the following theorem.

▶ **Theorem 13.** *There exists a countable, finitely branching and acyclic MDP $\mathcal{M}$ whose step counter is implicit in the state for which s_0 is almost surely winning $MP_{\liminf\geq 0}$, i.e., $\exists\hat{\sigma}\, \mathcal{P}_{\mathcal{M},s_0,\hat{\sigma}}(MP_{\liminf\geq 0}) = 1$, but every FR strategy σ is such that $\mathcal{P}_{\mathcal{M},s_0,\sigma}(MP_{\liminf\geq 0}) = 0$. In particular, almost sure winning strategies, when they exist, cannot be chosen k-bit Markov for any $k \in \mathbb{N}$ for countable MDPs.*

All of the above results/proofs also hold for $TP_{\liminf\geq 0}$, giving us the following theorem.

▶ **Theorem 14.** *There exists a countable, finitely branching and acyclic MDP $\mathcal{M}$ whose step counter is implicit in the state for which s_0 is almost surely winning $TP_{\liminf\geq 0}$, i.e., $\exists\hat{\sigma}\, \mathcal{P}_{\mathcal{M},s_0,\hat{\sigma}}(TP_{\liminf\geq 0}) = 1$, but every FR strategy σ is such that $\mathcal{P}_{\mathcal{M},s_0,\sigma}(TP_{\liminf\geq 0}) = 0$. In particular, almost sure winning strategies, when they exist, cannot be chosen k-bit Markov for any $k \in \mathbb{N}$ for countable MDPs.*

4 When is a reward counter not sufficient?

In this part we show that a reward counter plus arbitrary finite memory does not suffice for (ε-)optimal strategies for $MP_{\liminf \geq 0}$, even if the MDP is finitely branching.

The same lower bound holds for $TP_{\liminf \geq 0}/PP_{\liminf \geq 0}$, but only in infinitely branching MDPs. The finitely branching case is different for $TP_{\liminf \geq 0}/PP_{\liminf \geq 0}$; cf. Section 5.

The techniques used to prove these results are similar to those in Section 3 and proofs can be found in [17].

▶ **Theorem 15.** *There exists a countable, finitely branching, acyclic MDP $\mathcal{M}_{\mathrm{RI}}$ with initial state $(s_0, 0)$ with the total reward implicit in the state such that*

- $\mathtt{val}_{\mathcal{M}_{\mathrm{RI}}, MP_{\liminf \geq 0}}((s_0, 0)) = 1$,
- *for all FR strategies σ, we have $\mathcal{P}_{\mathcal{M}_{\mathrm{RI}},(s_0,0),\sigma}(MP_{\liminf \geq 0}) = 0$.*

▶ **Theorem 16.** *There exists a countable, finitely branching and acyclic MDP $\mathcal{M}_{\mathrm{Restart}}$ whose total reward is implicit in the state where, for the initial state s_0,*

- *there exists an HD strategy σ s.t. $\mathcal{P}_{\mathcal{M}_{\mathrm{Restart}},s_0,\sigma}(MP_{\liminf \geq 0}) = 1$.*
- *for every FR strategy σ, $\mathcal{P}_{\mathcal{M}_{\mathrm{Restart}},s_0,\sigma}(MP_{\liminf \geq 0}) = 0$.*

▶ **Theorem 17.** *There exists an infinitely branching MDP $\mathcal{M}$ with reward implicit in the state and initial state s such that*

- *every FR strategy σ is such that $\mathcal{P}_{\mathcal{M},s,\sigma}(TP_{\liminf \geq 0}) = 0$ and $\mathcal{P}_{\mathcal{M},s,\sigma}(PP_{\liminf \geq 0}) = 0$*
- *there exists an HD strategy σ s.t. $\mathcal{P}_{\mathcal{M},s,\sigma}(TP_{\liminf \geq 0}) = 1$ and $\mathcal{P}_{\mathcal{M},s,\sigma}(PP_{\liminf \geq 0}) = 1$.*

Hence, optimal (and even almost-surely winning) strategies and ε-optimal strategies for $TP_{\liminf \geq 0}$ and $PP_{\liminf \geq 0}$ require infinite memory beyond a reward counter.

▶ Remark 18. The MDPs from Section 3 and Section 4 show that good strategies for $MP_{\liminf \geq 0}$ require at least (in the sense of Remark 1) a reward counter and a step counter, respectively. There does, of course, exist a *single MDP* where good strategies for $MP_{\liminf \geq 0}$ require at least both a step counter and a reward counter. We construct such an MDP by "gluing" the two different MDPs together via an initial random state which points to each with probability 1/2.

5 Upper bounds

We establish upper bounds on the strategy complexity of lim inf threshold objectives for mean payoff, total payoff and point payoff. It is noteworthy that once the reward structure of an MDP has been encoded into the states, then these threshold objectives take on a qualitative flavor not dissimilar to Safety or co-Büchi (cf. [16]). Indeed, if the transition rewards are restricted to integer values, then $TP_{\liminf \geq 0}$ boils down to eventually avoiding all transitions with negative reward (since negative rewards would be ≤ -1). This is a co-Büchi objective. However, if the rewards are not restricted to integers, then the picture is not so simple.

For *finitely branching* MDPs, we show that there exist ε-optimal MD strategies for $PP_{\liminf \geq 0}$. In turn, this yields the requisite upper bound for finitely branching $TP_{\liminf \geq 0}$, i.e., using just a reward counter.

For *infinitely branching* MDPs, a step counter suffices in order to achieve $PP_{\liminf \geq 0}$ ε-optimally. Then, by encoding the total reward into the states, this will also give us SC+RC upper bounds for $MP_{\liminf \geq 0}$ as well as infinitely branching $TP_{\liminf \geq 0}$ (i.e., using both a step counter and a reward counter).

First we show how to encode the total reward level into the state in a given MDP.

▶ Remark 19. Given an MDP $\mathcal{M}$ and initial state s_0, we can construct an MDP $R(\mathcal{M})$ with initial state $(s_0, 0)$ and with the reward counter implicit in the state such that strategies in $R(\mathcal{M})$ can be translated back to $\mathcal{M}$ with an extra reward counter.

By labeling transitions in $R(\mathcal{M})$ with the state encoded total reward of the target state, we ensure that the point rewards in $R(\mathcal{M})$ correspond exactly to the total rewards in $\mathcal{M}$.

▶ **Lemma 20.** *Let $\mathcal{M}$ be an MDP with initial state s_0. Then given an MD (resp. Markov) strategy σ' in $R(\mathcal{M})$ attaining $c \in [0, 1]$ for $PP_{\liminf \geq 0}$ from $(s_0, 0)$, there exists a strategy σ attaining c for $TP_{\liminf \geq 0}$ in $\mathcal{M}$ from s_0 which uses the same memory as σ' plus a reward counter.*

▶ Remark 21. Given an MDP $\mathcal{M}$ and initial state s_0, we can construct an acyclic MDP $S(\mathcal{M})$ with initial state $(s_0, 0)$ and with the step counter implicit in the state such that MD strategies in $S(\mathcal{M})$ can be translated back to $\mathcal{M}$ with the use of a step counter to yield deterministic Markov strategies in $\mathcal{M}$; cf. [15, Lemma 4].

▶ Remark 22. In order to tackle the mean payoff objective $MP_{\liminf \geq 0}$ on $\mathcal{M}$, we define a new acyclic MDP $A(\mathcal{M})$ which encodes both the step counter and the average reward into the state. However, since we want the point rewards in $A(\mathcal{M})$ to coincide with the *mean payoff* in the original MDP $\mathcal{M}$, the transition rewards in $A(\mathcal{M})$ are given as the encoded rewards divided by the step counter (unlike in $R(\mathcal{M})$).

▶ **Lemma 23.** *Let $\mathcal{M}$ be an MDP with initial state s_0. Then given an MD strategy σ' in $A(\mathcal{M})$ attaining $c \in [0, 1]$ for $PP_{\liminf \geq 0}$ from $(s_0, 0, 0)$, there exists a strategy σ attaining c for $MP_{\liminf \geq 0}$ in $\mathcal{M}$ from s_0 which uses just a reward counter and a step counter.*

Proof. The proof is very similar to that of Lemma 20. ◀

▶ **Lemma 24** ([15, Lemma 23])**.** *For every acyclic MDP with a safety objective and every $\varepsilon > 0$, there exists an MD strategy that is uniformly ε-optimal.*

▶ **Theorem 25** ([13, Theorem 7])**.** *Let $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P, r)$ be a countable MDP, and let φ be an event that is tail in $\mathcal{M}$. Suppose for every $s \in S$ there exist ε-optimal MD strategies for φ. Then:*

1. *There exist uniform ε-optimal MD strategies for φ.*
2. *There exists a single MD strategy that is optimal from every state that has an optimal strategy.*

5.1 Finitely Branching Case

In order to prove the main result of this section, we use the following result on the `Transience` objective, which is the set of runs that do not visit any state infinitely often. Given an MDP $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P, r)$, $\texttt{Transience} \stackrel{\text{def}}{=} \bigwedge_{s \in S} \mathsf{F}\mathsf{G}\neg s$.

▶ **Theorem 26** ([13, Theorem 8])**.** *In every countable MDP there exist uniform ε-optimal MD strategies for* `Transience`.

▶ **Theorem 27.** *Consider a finitely branching MDP $\mathcal{M} = (S, S_\Box, S_\bigcirc, \longrightarrow, P, r)$ with initial state s_0 and a $PP_{\liminf \geq 0}$ objective. Then there exist ε-optimal MD strategies.*

Proof. Let $\varepsilon > 0$. We begin by partitioning the state space into two sets, S_{safe} and $S \setminus S_{\text{safe}}$. The set S_{safe} is the subset of states which is surely winning for the safety objective of only using transitions with non-negative rewards (i.e., never using transitions with negative rewards at all). Since $\mathcal{M}$ is finitely branching, there exists a uniformly optimal MD strategy σ_{safe} for this safety objective [18, 16].

We construct a new MDP $\mathcal{M}'$ by modifying $\mathcal{M}$. We create a gadget G_{safe} composed of a sequence of new controlled states $x_0, x_1, x_2, \ldots$ where all transitions $x_i \to x_{i+1}$ have reward 0. Hence any run entering G_{safe} is winning for $PP_{\liminf\geq 0}$. We insert G_{safe} into $\mathcal{M}$ by replacing all incoming transitions to S_{safe} with transitions that lead to x_0. The idea behind this construction is that when playing in $\mathcal{M}$, once you hit a state in S_{safe}, you can win surely by playing an optimal MD strategy for safety. So we replace S_{safe} with the surely winning gadget G_{safe}. Thus

$$\mathtt{val}_{\mathcal{M},PP_{\liminf\geq 0}}(s_0) = \mathtt{val}_{\mathcal{M}',PP_{\liminf\geq 0}}(s_0) \tag{1}$$

and if an ε-optimal MD strategy exists in $\mathcal{M}$, then there exists a corresponding one in $\mathcal{M}'$, and vice-versa.

We now consider a general (not necessarily MD) ε-optimal strategy σ for $PP_{\liminf\geq 0}$ from s_0 on $\mathcal{M}'$, i.e.,

$$\mathcal{P}_{\mathcal{M}',s_0,\sigma}(PP_{\liminf\geq 0}) \geq \mathtt{val}_{\mathcal{M}',PP_{\liminf\geq 0}}(s_0) - \varepsilon. \tag{2}$$

Define the safety objective Safety_i which is the objective of never seeing any point rewards $< -2^{-i}$. This then allows us to characterize $PP_{\liminf\geq 0}$ in terms of safety objectives.

$$PP_{\liminf\geq 0} = \bigcap_{i\in\mathbb{N}} \mathsf{F}(\text{Safety}_i). \tag{3}$$

Now we define the safety objective $\text{Safety}_i^k \stackrel{\text{def}}{=} \mathsf{F}^{\leq k}(\text{Safety}_i)$ to attain Safety_i within at most k steps. This allows us to write

$$\mathsf{F}(\text{Safety}_i) = \bigcup_{k\in\mathbb{N}} \text{Safety}_i^k. \tag{4}$$

By continuity of measures from above we get

$$0 = \mathcal{P}_{\mathcal{M}',s_0,\sigma}\left(\mathsf{F}(\text{Safety}_i) \cap \bigcap_{k\in\mathbb{N}} \overline{\text{Safety}_i^k}\right) = \lim_{k\to\infty} \mathcal{P}_{\mathcal{M}',s_0,\sigma}\left(\mathsf{F}(\text{Safety}_i) \cap \overline{\text{Safety}_i^k}\right).$$

Hence for every $i \in \mathbb{N}$ and $\varepsilon_i \stackrel{\text{def}}{=} \varepsilon \cdot 2^{-i}$ there exists n_i such that

$$\mathcal{P}_{\mathcal{M}',s_0,\sigma}\left(\mathsf{F}(\text{Safety}_i) \cap \overline{\text{Safety}_i^{n_i}}\right) \leq \varepsilon_i. \tag{5}$$

Now we can show the following claim (proof in [17]).

▷ Claim 28.

$$\mathcal{P}_{\mathcal{M}',s_0,\sigma}\left(\bigcap_{i\in\mathbb{N}} \text{Safety}_i^{n_i}\right) \geq \mathtt{val}_{\mathcal{M}',PP_{\liminf\geq 0}}(s_0) - 2\varepsilon.$$

Since $\mathcal{M}'$ does not have an implicit step counter, we use the following construction to approximate one. We define the distance $d(s)$ from s_0 to a state s as the length of the shortest path from s_0 to s. Let $\text{Bubble}_n(s_0) \stackrel{\text{def}}{=} \{s \in S \mid d(s) \leq n\}$ be those states that can be reached within n steps from s_0. Since $\mathcal{M}'$ is finitely branching, $\text{Bubble}_n(s_0)$ is finite for every fixed n. Let

$$\text{Bad}_i \stackrel{\text{def}}{=} \{t \in \longrightarrow_{\mathcal{M}'} \mid t = s \longrightarrow_{\mathcal{M}'} s', s \notin \text{Bubble}_{n_i}(s_0) \text{ and } r(t) < -2^{-i}\}$$

be the set of transitions originating outside $\text{Bubble}_{n_i}(s_0)$ whose reward is too negative. Thus a run from s_0 that satisfies $\text{Safety}_i^{n_i}$ cannot use any transition in Bad_i, since (by definition of $\text{Bubble}_{n_i}(s_0)$) they would come after the n_i-th step.

Now we create a new state $\perp$ whose only outgoing transition is a self loop with reward -1. We transform $\mathcal{M}'$ into $\mathcal{M}''$ by re-directing all transitions in Bad_i to the new target state $\perp$ for every i. Notice that any run visiting $\perp$ must be losing for $PP_{\liminf\geq 0}$ due to the negative reward on the self loop, but it must also be losing for `Transience` because of the self loop.

We now show that the change from $\mathcal{M}'$ to $\mathcal{M}''$ has decreased the value of s_0 for $PP_{\liminf\geq 0}$ by at most 2ε, i.e.,

$$\mathtt{val}_{\mathcal{M}'',PP_{\liminf\geq 0}}(s_0) \geq \mathtt{val}_{\mathcal{M}',PP_{\liminf\geq 0}}(s_0) - 2\varepsilon. \tag{6}$$

Equation (6) follows from the following steps.

$$\begin{aligned}
\mathtt{val}_{\mathcal{M}'',PP_{\liminf\geq 0}}(s_0) &\geq \mathcal{P}_{\mathcal{M}'',s_0,\sigma}\left(\bigcap_{i\in\mathbb{N}} \text{Safety}_i^{n_i}\right) \\
&= \mathcal{P}_{\mathcal{M}',s_0,\sigma}\left(\bigcap_{i\in\mathbb{N}} \text{Safety}_i^{n_i}\right) && \text{by def. of } \mathcal{M}'' \\
&\geq \mathtt{val}_{\mathcal{M}',PP_{\liminf\geq 0}}(s_0) - 2\varepsilon && \text{by Claim 28}
\end{aligned}$$

In the next step (proof in [17]) we argue that under *every* strategy σ'' from s_0 in $\mathcal{M}''$ the attainment for $PP_{\liminf\geq 0}$ and `Transience` coincide, i.e.,

▷ Claim 29.

$$\forall\sigma''.\, \mathcal{P}_{\mathcal{M}'',s_0,\sigma''}(PP_{\liminf\geq 0}) = \mathcal{P}_{\mathcal{M}'',s_0,\sigma''}(\mathtt{Transience}).$$

By Theorem 26, there exists a uniformly ε-optimal MD strategy $\widehat{\sigma}$ from s_0 for `Transience` in $\mathcal{M}''$, i.e.,

$$\mathcal{P}_{\mathcal{M}'',s_0,\hat{\sigma}}(\mathtt{Transience}) \geq \mathtt{val}_{\mathcal{M}'',\mathtt{Transience}}(s_0) - \varepsilon. \tag{7}$$

We construct an MD strategy σ^* in $\mathcal{M}$ which plays like σ_{safe} in S_{safe} and plays like $\widehat{\sigma}$ everywhere else.

$$\begin{aligned}
\mathcal{P}_{\mathcal{M},s_0,\sigma^*}(PP_{\liminf\geq 0}) &= \mathcal{P}_{\mathcal{M}',s_0,\hat{\sigma}}(PP_{\liminf\geq 0}) && \text{def. of } \sigma^* \text{ and } \sigma_{\text{safe}} \\
&\geq \mathcal{P}_{\mathcal{M}'',s_0,\hat{\sigma}}(PP_{\liminf\geq 0}) && \text{new losing sink in } \mathcal{M}'' \\
&= \mathcal{P}_{\mathcal{M}'',s_0,\hat{\sigma}}(\mathtt{Transience}) && \text{by Claim 29} \\
&\geq \mathtt{val}_{\mathcal{M}'',\mathtt{Transience}}(s_0) - \varepsilon && \text{by (7)} \\
&= \mathtt{val}_{\mathcal{M}'',PP_{\liminf\geq 0}}(s_0) - \varepsilon && \text{by Claim 29} \\
&\geq \mathtt{val}_{\mathcal{M}',PP_{\liminf\geq 0}}(s_0) - 2\varepsilon - \varepsilon && \text{by (6)} \\
&= \mathtt{val}_{\mathcal{M},PP_{\liminf\geq 0}}(s_0) - 3\varepsilon && \text{by (1)}
\end{aligned}$$

Hence σ^* is a 3ε-optimal MD strategy for $PP_{\liminf\geq 0}$ from s_0 in $\mathcal{M}$ as required. ◀

▶ **Corollary 30.** *Given a finitely branching MDP $\mathcal{M}$, there exist ε-optimal strategies for $TP_{\liminf\geq 0}$ which use just a reward counter.*

Proof. By Theorem 27 and Lemma 20. ◀

▶ **Corollary 31.** *Given a finitely branching MDP $\mathcal{M}$ and initial state s_0, optimal strategies, where they exist,*

- *for $PP_{\liminf\geq 0}$ can be chosen MD.*
- *for $TP_{\liminf\geq 0}$ can be chosen with just a reward counter.*

Proof. Since $PP_{\liminf\geq 0}$ is tail, the first claim follows from Theorem 27 and Theorem 25.

Towards the second claim, we place ourselves in $R(\mathcal{M})$ where $TP_{\liminf\geq 0}$ is tail. Moreover, in $R(\mathcal{M})$ the objectives $TP_{\liminf\geq 0}$ and $PP_{\liminf\geq 0}$ coincide. Thus we can apply Theorem 27 to obtain ε-optimal MD strategies for $TP_{\liminf\geq 0}$ from every state of $R(\mathcal{M})$. From Theorem 25 we obtain a single MD strategy that is optimal from every state of $R(\mathcal{M})$ that has an optimal strategy. By Lemma 20 we can translate this MD strategy on $R(\mathcal{M})$ back to a strategy on $\mathcal{M}$ with just a reward counter. ◀

5.2 Infinitely Branching Case

For infinitely branching MDPs, ε-optimal strategies for $PP_{\liminf\geq 0}$ require more memory than in the finitely branching case. However, the proofs are similar to those in Section 5.1 and can be found in [17].

▶ **Theorem 32.** *Consider an MDP $\mathcal{M}$ with initial state s_0 and a $PP_{\liminf\geq 0}$ objective. For every $\varepsilon > 0$ there exist*

- *ε-optimal MD strategies in $S(\mathcal{M})$.*
- *ε-optimal deterministic Markov strategies in $\mathcal{M}$.*

▶ **Corollary 33.** *Given an MDP $\mathcal{M}$ and initial state s_0, there exist ε-optimal strategies σ for $MP_{\liminf\geq 0}$ which use just a step counter and a reward counter.*

▶ **Corollary 34.** *Given an MDP $\mathcal{M}$ with initial state s_0,*

- *there exist ε-optimal MD strategies for $TP_{\liminf\geq 0}$ in $S(R(\mathcal{M}))$,*
- *there exist ε-optimal strategies for $TP_{\liminf\geq 0}$ which use a step counter and a reward counter.*

▶ **Corollary 35.** *Given an MDP $\mathcal{M}$ and initial state s_0, optimal strategies, where they exist,*

- *for $PP_{\liminf\geq 0}$ can be chosen with just a step counter.*
- *for $MP_{\liminf\geq 0}$ and $TP_{\liminf\geq 0}$ can be chosen with just a reward counter and a step counter.*

6 Conclusion and Outlook

We have established matching lower and upper bounds on the strategy complexity of $\liminf$ threshold objectives for point, total and mean payoff on countably infinite MDPs; cf. Table 1.

The upper bounds hold not only for integer transition rewards, but also for rationals or reals, provided that the reward counter (in those cases where one is required) is of the same type. The lower bounds hold even for integer transition rewards, since all our counterexamples are of this form.

Directions for future work include the corresponding questions for $\limsup$ threshold objectives. While the $\liminf$ point payoff objective generalizes co-Büchi (see Section 2), the $\limsup$ point payoff objective generalizes Büchi. Thus the lower bounds for $\limsup$ point payoff are at least as high as the lower bounds for Büchi objectives [14, 15].

References

1 Pieter Abbeel and Andrew Y. Ng. Learning first-order Markov models for control. In *Advances in Neural Information Processing Systems 17*, pages 1–8. MIT Press, 2004. URL: `http://papers.nips.cc/paper/2569-learning-first-order-markov-models-for-control`.

2 Galit Ashkenazi-Golan, János Flesch, Arkadi Predtetchinski, and Eilon Solan. Reachability and safety objectives in Markov decision processes on long but finite horizons. *Journal of Optimization Theory and Applications*, 185:945–965, 2020.

3 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

4 P. Billingsley. *Probability and Measure*. Wiley, New York, NY, 1995. Third Edition.

5 Vincent D. Blondel and John N. Tsitsiklis. A survey of computational complexity results in systems and control. *Automatica*, 36(9):1249–1274, 2000.

6 Nicole Bäuerle and Ulrich Rieder. *Markov Decision Processes with Applications to Finance*. Springer-Verlag Berlin Heidelberg, 2011.

7 K. Chatterjee, L. Doyen, and T. Henzinger. A survey of stochastic games with limsup and liminf objectives. In *Proc. of ICALP*, volume 5556 of *LNCS*. Springer, 2009.

8 Krishnendu Chatterjee and Laurent Doyen. Games and Markov decision processes with mean-payoff parity and energy parity objectives. In *Proc. of MEMICS*, volume 7119 of *LNCS*, pages 37–46. Springer, 2011.

9 Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors. *Handbook of Model Checking*. Springer, 2018. `doi:10.1007/978-3-319-10575-8`.

10 E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, December 1999.

11 János Flesch, Arkadi Predtetchinski, and William Sudderth. Simplifying optimal strategies in limsup and liminf stochastic games. *Discrete Applied Mathematics*, 251:40–56, 2018.

12 T.P. Hill and V.C. Pestien. The existence of good Markov strategies for decision processes with general payoffs. *Stoch. Processes and Appl.*, 24:61–76, 1987.

13 S. Kiefer, R. Mayr, M. Shirmohammadi, and P. Totzke. Transience in countable MDPs. In *Proc. of CONCUR*, volume 203 of *LIPIcs*, 2021. Full version at `arXiv:2012.13739`.

14 Stefan Kiefer, Richard Mayr, Mahsa Shirmohammadi, and Patrick Totzke. Büchi objectives in countable MDPs. In *ICALP*, volume 132 of *LIPIcs*, pages 119:1–119:14, 2019. Full version at `arXiv:1904.11573`. `doi:10.4230/LIPIcs.ICALP.2019.119`.

15 Stefan Kiefer, Richard Mayr, Mahsa Shirmohammadi, and Patrick Totzke. Strategy Complexity of Parity Objectives in Countable MDPs. In *CONCUR*, pages 7:1–:17, 2020. `doi:10.4230/LIPIcs.CONCUR.2020.7`.

16 Stefan Kiefer, Richard Mayr, Mahsa Shirmohammadi, and Dominik Wojtczak. Parity Objectives in Countable MDPs. In *LICS*. IEEE, 2017. `doi:10.1109/LICS.2017.8005100`.

17 Richard Mayr and Eric Munday. Strategy Complexity of Mean Payoff, Total Payoff and Point Payoff Objectives in Countable MDPs. In *Proc. of CONCUR*, volume 203 of *LIPIcs*, 2021. Full version at `arXiv:2107.03287`.

18 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

19 S. M. Ross. *Introduction to Stochastic Dynamic Programming*. Academic Press, New York, 1983.

20 Manfred Schäl. Markov decision processes in finance and dynamic options. In *Handbook of Markov Decision Processes*, pages 461–487. Springer, 2002.

21 Olivier Sigaud and Olivier Buffet. *Markov Decision Processes in Artificial Intelligence*. John Wiley & Sons, 2013.

22 William D. Sudderth. Optimal Markov strategies. *Decisions in Economics and Finance*, 43:43–54, 2020.

23 R.S. Sutton and A.G Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. MIT Press, 2018.

24 M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proc. of FOCS'85*, pages 327–338, 1985.

Model Checking Quantum Continuous-Time Markov Chains

Ming Xu
Shanghai Key Lab of Trustworthy Computing,
MoE Engineering Research Center of Software/Hardware Co-design Technology and Application,
East China Normal University, Shanghai, China

Jingyi Mei
Shanghai Key Lab of Trustworthy Computing, East China Normal University, Shanghai, China

Ji Guan
State Key Lab of Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing, China

Nengkun Yu
Centre for Quantum Software and Information, Faculty of Engineering and Information Technology,
University of Technology, Sydney, Australia

Abstract

Verifying quantum systems has attracted a lot of interests in the last decades. In this paper, we initialise the model checking of quantum continuous-time Markov chain (QCTMC). As a real-time system, we specify the temporal properties on QCTMC by signal temporal logic (STL). To effectively check the atomic propositions in STL, we develop a state-of-the-art real root isolation algorithm under Schanuel's conjecture; further, we check the general STL formula by interval operations with a bottom-up fashion, whose query complexity turns out to be linear in the size of the input formula by calling the real root isolation algorithm. A running example of an open quantum walk is provided to demonstrate our method.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Computing methodologies → Symbolic and algebraic algorithms; Theory of computation → Quantum computation theory

Keywords and phrases Model Checking, Formal Logic, Quantum Computing, Computer Algebra

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.13

Related Version *Previous Version*: https://arxiv.org/abs/2105.00382

Funding *Ming Xu*: The corresponding author M. X. is supported by the National Natural Science Foundation of China (No. 11871221), the National Key R&D Program of China (No. 2018YFA0306704), the Fundamental Research Funds for the Central Universities, and the Research Funds of Happiness Flower ECNU (No. 2020ECNU-XFZH005).
Ji Guan: The corresponding author J. G. is supported by the National Key R&D Program of China (No. 2018YFA0306701) and the National Natural Science Foundation of China (No. 61832015).
Nengkun Yu: N. Y. is supported by ARC Discovery Early Career Researcher Award (No. DE180100156) and ARC Discovery Program (No. DP210102449).

1 Introduction

Aiming to study nature, physicists use different mechanics depending on the scale of the objects they are interested in. Classical mechanics describes nature at a macroscopic scale (far larger than 10^{-9} meters), while quantum mechanics is applied at a microscopic scale (near or less than 10^{-9} meters).

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 13; pp. 13:1–13:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In physics, a *closed* quantum system is a physical system that does not interchange information (i.e. energy and/or matter) with another system. The behavior of a closed quantum system can be described as follows: A particle at this level can be mathematically represented by a normalised complex vector $|s\rangle$ in a Hilbert space $\mathcal{H}$. The time evolution of a single particle *closed* system is described by the Schrödinger equation:

$$\frac{\mathrm{d}\,|s(t)\rangle}{\mathrm{d}t} = -\imath\mathbf{H}\,|s(t)\rangle \tag{1}$$

with some *Hamiltonian* $\mathbf{H}$ (a Hermitian matrix on $\mathcal{H}$), where $|s(t)\rangle$ is the state of the system at time t.

More practically, an *open* quantum system interacting with the surrounding environment need to be considered. An open quantum system is a quantum-mechanical system that interacts with an external quantum system, which is known as the environment or a bath. In general, these interactions significantly change the dynamics of the system and result in quantum dissipation, such that the information contained in the system is lost to its environment. Suffering noises from the environment, the state of the system cannot be completely known. Thus a *density operator* ρ (positive semidefinite matrix with unit trace) on $\mathcal{H}$ is introduced to describe the uncertainty of the possible states: $\rho = \sum_{i\in I} p_i\,|s_i\rangle\langle s_i|$, where $\{(p_i, |s_i\rangle)\}_{i\in I}$ is a mixed state or an ensemble expressing that the quantum state is at $|s_i\rangle$ with probability p_i, and $\langle s_i|$ is the complex conjugate and transpose of $|s_i\rangle$. In this case, the evolution is described by the Lindblad's master equation:

$$\frac{\mathrm{d}\rho(t)}{\mathrm{d}t} = \mathcal{L}(\rho(t)) \tag{2}$$

where $\rho(t)$ stands for the (possibly mixed) state of the system, and $\mathcal{L}$ is a linear function of $\rho(t)$ (to be formally described in Subsection 2.2), which is generally irreversible.

To reveal physical phenomenon, physicists have intensively studied the properties of closed and open quantum systems case by case in the last decades, such as long-term behaviors (e.g. [18]) and stabilities (e.g. [10]). In recent years, the computer science community has stepped into this field and adopted model checking technique to study quantum systems [16, 17]. Specifically, the quantum systems can be simulated by some mathematical models and a bulk of physical properties can be reformulated as formulas in some temporal logic with atomic propositions of quantum interpretation. In particular, a *quantum discrete-time Markov chain* (QDTMC) $(\mathcal{H}, \mathcal{E})$ has been introduced as quantum generalisations of classical discrete-time Markov chains (DTMC) to model the evolution in Eq. (2) in a single time unit:

$$\rho(t+1) = \mathcal{E}(\rho(t)) \tag{3}$$

where $\mathcal{E}$ is a discretised quantum operation obtained from the Lindblad's master equation, usually called a *super-operator* in the field of quantum information and computation. Several fundamental model checking-related problems for QDTMCs have been studied in the literature, including limiting states [40, 21], reachability [45, 41], repeated reachability [14], linear time properties [30], and persistence based on irreducible and periodic decomposition techniques [8, 21]. These techniques were equipped to solve real-world problems in several different areas. For example, [15] proposed algorithms to model check quantum cryptographic protocols against a quantum extension of probabilistic computation tree logic (PCTL); and linear temporal logic (LTL) was adopted to specify a bulk of properties of quantum many-body and statistical systems, and corresponding model checking algorithm was developed [20]. For quantum concurrent systems, the reachability and the termination are studied in [47]. Very recently, Yu investigated a quantum temporal logic for the specification of the system and studied the model checking problem for basic formulas [46]. See [43, 44] for the comprehensive review of this research line.

However, to the best of our knowledge, there is no work on model checking the quantum continuous-time system of Eq. (2). In contrast, there are fruitful results in the classical counterpart, which is usually modelled by a continuous-time Markov chain (CTMC).

The seminal work on verifying CTMCs is Aziz *et al.*'s paper [3, 4]. The authors introduced continuous stochastic logic (CSL) interpreted on CTMCs. Roughly speaking, the syntax of CSL amounts to that of PCTL plus the multiphase until formula $\Phi_1 \mathrm{U}^{\mathcal{I}_1} \Phi_2 \cdots \mathrm{U}^{\mathcal{I}_K} \Phi_{K+1}$, for some $K \geq 1$. Because [3] restricts probabilities in $\mathrm{Pr}_{>\mathtt{c}}$ to $\mathtt{c} \in \mathbb{Q}$, they can show decidability of model checking for CSL using number-theoretic analysis. An approximate model checking algorithm for a reduced version of CSL was provided by Baier *et al.* [6], who restrict path formulas to binary until: $\Phi_1 \mathrm{U}^{\mathcal{I}} \Phi_2$. Under this logic, they successfully applied efficient numerical techniques for *transient analysis* [5] using *uniformisation* [37]. The approximate algorithms have been extended for multiphase until formulas using *stratification* [48, 49]. Xu *et al.* considered the multi-phase until formulas over the CTMC with rewards [42]. An integral-style algorithm was proposed to attack this problem, whose effectiveness is ensured by number-theoretic results and algebraic methods. Recently, continuous linear logic was introduced to specify on CTMCS, whose decidability was established [23]. Most the above algorithms have been implemented in probabilistic model checkers, like *PRISM* [29], *Storm* [13], and *ePMC* [24].

Unfortunately, these results from CTMCs cannot directly tackle the problem of automatically verifying quantum continuous-time systems. The main obstacle is that the state space of classical case is finite, while the space of quantum states in even a finite-dimensional Hilbert space is a continuum. Even though the probability space of a CTMC is also a continuum, a probability distribution can only represent one ensemble (i.e. the uncertainty of the possible states), while a quantum state generally admits infinitely many ensembles, such as $\frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1| = \frac{1}{2}|\psi_1\rangle\langle\psi_1| + \frac{1}{2}|\psi_2\rangle\langle\psi_2|$ for any two orthogonal normalised complex vectors $|\psi_1\rangle$ and $|\psi_2\rangle$ on the Hilbert space linearly spanned by orthonormal basis $\{|0\rangle, |1\rangle\}$. In this paper, we cross the difficulty by introducing quantum continuous-time Markov chains (QCTMC) to model the evolution of quantum continuous-time systems in Eq. (2) and converting them into a distribution transformer that preserves the laws of quantum mechanics. Then, we consider a wide logic, signal temporal logic (STL), to specify real-time properties against QCTMC. The STL is more expressible than LTL and CTL. Finally we present an exact method to decide the STL formula using real root isolation and interval operations, whose query complexity turns out to be linear in the size of the input formula by calling the real root isolation routine.

The key contributions of the present paper are three-fold:

1. In the field of formal verification, the model checking on DTMC, QDTMC and CTMC has been well studied in the past decades, but no work on checking QCTMCs as far as we know. The first contribution is filling this blank.
2. In order to solve the atomic propositions in STL, we develop a state-of-the-art real root isolation algorithm for a rich class of real-valued functions based on Schanuel's conjecture.
3. We provide a running example – open quantum walk equipped with an absorbing boundary, which drops the restriction on the underlying graph being symmetric/Hermitian. The non-Hermitian structure brings real technical hardness. Fortunately, it is overcome by Eq. (2) employed for describing the dynamical system of QCTMC.

Organisation. The rest of the paper is structured as follows. Section 2 reviews some notions and notations from quantum computing, the Lindblad's master equation and number theory. In Sections 3 and 4, we introduce the model of quantum continuous-time Markov chains and the signal temporal logic (STL), respectively. We solve the atomic propositions in STL in Section 5, and decide the general STL formulas in Section 6. Section 7 is the conclusion.

2 Preliminaries

2.1 Quantum Computing

Let $\mathcal{H}$ be a Hilbert space with dimension d. We employ the Dirac notations that are standard in quantum computing:

- $|s\rangle$ stands for a unit column vector in $\mathcal{H}$ labelled with s;
- $\langle s| := |s\rangle^\dagger$ is the Hermitian adjoint (complex conjugate and transpose) of $|s\rangle$;
- $\langle s_1|s_2\rangle := \langle s_1|\,|s_2\rangle$ is the inner product of $|s_1\rangle$ and $|s_2\rangle$;
- $|s_1\rangle\langle s_2| := |s_1\rangle \otimes \langle s_2|$ is the outer product, where $\otimes$ denotes tensor product; and
- $|s_1, s_2\rangle := |s_1\rangle\,|s_2\rangle$ is a shorthand of the product state $|s_1\rangle \otimes |s_2\rangle$.

A linear operator γ is *Hermitian* if $\gamma = \gamma^\dagger$; and it is *positive* if $\langle s|\,\gamma\,|s\rangle \geq 0$ holds for all $|s\rangle \in \mathcal{H}$. A *projector* $\mathbf{P}$ is a positive operator of the form $\sum_{i=1}^m |s_i\rangle\langle s_i|$ with $m \leq d$, where $|s_i\rangle$ are orthonormal. Clearly, there is a bijective map between projectors $\mathbf{P} = \sum_{i=1}^m |s_i\rangle\langle s_i|$ and subspaces of $\mathcal{H}$ that are spanned by $\{|s_i\rangle : 1 \leq i \leq m\}$. In sum, positive operators are Hermitian ones whose eigenvalues are nonnegative; and projectors are positive operators whose eigenvalues are 0 or 1. Besides, a linear operator $\mathbf{U}$ is *unitary* if $\mathbf{U}\mathbf{U}^\dagger = \mathbf{U}^\dagger\mathbf{U} = \mathbf{I}$ where $\mathbf{I}$ is the *identity* operator.

The *trace* of a linear operator γ is defined as $\operatorname{tr}(\gamma) := \sum_{i=1}^d \langle s_i|\,\gamma\,|s_i\rangle$ for any orthonormal basis $\{|s_i\rangle : 1 \leq i \leq d\}$ of $\mathcal{H}$. A *density operator* ρ on $\mathcal{H}$ is a positive operator with unit trace. It gives rise to a generic way to describe quantum states: if a density operator ρ is $|s\rangle\langle s|$ for some $|s\rangle \in \mathcal{H}$, ρ is said to be a *pure* state; otherwise it is a *mixed* one, i.e. $\rho = \sum_{i=1}^m p_i\,|s_i\rangle\langle s_i|$ with $m \geq 2$ by spectral decomposition, where p_i are positive eigenvalues (interpreted as the *probabilities* of taking the pure states $|s_i\rangle$) and their sum is 1. In other words, a pure state indicates the system state which we completely know; a mixed state gives all possible system states, with total probability 1, which we know. We denote by $\mathcal{D}_\mathcal{H}$ the set of density operators on $\mathcal{H}$. The subscript $\mathcal{H}$ of $\mathcal{D}_\mathcal{H}$ will be omitted if it is clear from the context.

The system evolution between pure states is characterised by some unitary operator $\mathbf{U}$, i.e. $|s(t)\rangle\langle s(t)| = \mathbf{U}(t)\,|s(0)\rangle\langle s(0)|\,\mathbf{U}^\dagger(t)$ where $\mathbf{U}(t)$ comes from $\exp(-\imath\mathbf{H}t)$ for the Hermitian operator $\mathbf{H}$ in Eq. (1); the system evolution between density operators (pure or mixed states) is characterised by some completely positive operator-sum, a.k.a. *super-operator*, i.e. $\rho(t) = \sum_{j=1}^m \mathbf{L}_j(t)\rho(0)\mathbf{L}_j^\dagger(t)$ where $\mathbf{L}_j$ are linear operators satisfying the trace preservation $\sum_{j=1}^m \mathbf{L}_j^\dagger\mathbf{L}_j = \mathbf{I}$. The latter dynamical system is obtained in such a way: an enlarged unitary operator acts on the purified composite state $(\sum_{i=1}^m \sqrt{p_i}\,|s_i, env_i\rangle)(\sum_{i=1}^m \sqrt{p_i}\,\langle s_i, env_i|)$ with $\rho(0) = \sum_{i=1}^m p_i\,|s_i\rangle\langle s_i|$ for some orthonormal environment states $|env_i\rangle$ [34, Section 2.5], then the environment in the resulting composite state is traced out, which turns out to be the aforementioned operator-sum form. We call it an *open* system as it interacts with the environment. Whereas, the former dynamical system, independent from the environment, is called a *closed* system. Open systems are more common than closed systems in practice.

2.2 Lindblad's Master Equation

To characterise state evolution of the continuous-time open system with the *memoryless* property, we employ the Lindblad's master equation [31, 19] that is

$$\rho' = \mathcal{L}(\rho) = -\imath \mathbf{H}\rho + \imath\rho\mathbf{H} + \sum_{j=1}^{m} \left(\mathbf{L}_j \rho \mathbf{L}_j^\dagger - \tfrac{1}{2}\mathbf{L}_j^\dagger \mathbf{L}_j \rho - \tfrac{1}{2}\rho \mathbf{L}_j^\dagger \mathbf{L}_j \right), \tag{4}$$

where $\mathbf{H}$ is a Hermitian operator and $\mathbf{L}_j$ are linear operators. The terms $-\imath\mathbf{H}\rho+\imath\rho\mathbf{H}$ describe the evolution of the internal system; the terms $\mathbf{L}_j\rho\mathbf{L}_j^\dagger - \frac{1}{2}\mathbf{L}_j^\dagger\mathbf{L}_j\rho - \frac{1}{2}\rho\mathbf{L}_j^\dagger\mathbf{L}_j$ describe the interaction between system and environment. In other words, to characterise the evolution of an open system, it is necessary to use those linear operators $\mathbf{L}_j$ besides the Hermitian operator $\mathbf{H}$. It is known to be the most general type of Markovian and time-homogeneous master equation describing (in general non-unitary) evolution of the system state that preserves the laws of quantum mechanics (i.e., completely positive and trace-preserving for any initial condition).

In the following, we will derive the solution of Eq. (4). We first define two useful functions:

- $\mathrm{L2V}(\gamma) := \sum_{i=1}^{d}\sum_{j=1}^{d} \langle i|\, \gamma\, |j\rangle\, |i,j\rangle$ that rearranges entries of the linear operator γ on the Hilbert space $\mathcal{H}$ with dimension d as a column vector; and
- $\mathrm{V2L}(\mathbf{v}) := \sum_{i=1}^{d}\sum_{j=1}^{d} \langle i,j|\, \mathbf{v}\, |i\rangle\langle j|$ that rearranges entries of the column vector $\mathbf{v}$ as a linear operator.

Here, L2V and V2L are read as "linear operator to vector" and "vector to linear operator", respectively. They are mutually inverse functions, so that if a linear operator (resp. its vectorisation) is determined, its vectorisation (resp. the original linear operator) is determined. Using the fact that $\mathbf{D} = \mathbf{ABC} \iff \mathrm{L2V}(\mathbf{D}) = (\mathbf{A}\otimes\mathbf{C}^{\mathrm{T}})\mathrm{L2V}(\mathbf{B})$ holds for any linear operators $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$, we can reformulate Eq. (4) as the linear ordinary differential equation

$$\begin{aligned} \mathrm{L2V}(\rho') &= \left[-\imath\mathbf{H}\otimes\mathbf{I} + \imath\mathbf{I}\otimes\mathbf{H}^{\mathrm{T}} + \sum_{j=1}^{m}\left(\mathbf{L}_j\otimes\mathbf{L}_j^* - \tfrac{1}{2}\mathbf{L}_j^\dagger\mathbf{L}_j\otimes\mathbf{I} - \tfrac{1}{2}\mathbf{I}\otimes\mathbf{L}_j^{\mathrm{T}}\mathbf{L}_j^* \right) \right] \mathrm{L2V}(\rho) \\ &= \mathbb{M}\cdot\mathrm{L2V}(\rho), \end{aligned} \tag{5}$$

where $*$ denotes entry-wise complex conjugate. We call $\mathbb{M} = -\imath\mathbf{H}\otimes\mathbf{I}+\imath\mathbf{I}\otimes\mathbf{H}^{\mathrm{T}}+\sum_{j=1}^{m}\left(\mathbf{L}_j\otimes\mathbf{L}_j^* - \frac{1}{2}\mathbf{L}_j^\dagger\mathbf{L}_j\otimes\mathbf{I} - \frac{1}{2}\mathbf{I}\otimes\mathbf{L}_j^{\mathrm{T}}\mathbf{L}_j^*\right)$ the *governing matrix* of Eq. (5). As a result, we get the desired solution $\mathrm{L2V}(\rho(t)) = \exp(\mathbb{M}\cdot t)\cdot\mathrm{L2V}(\rho(0))$ or equivalently $\rho(t) = \mathrm{V2L}(\exp(\mathbb{M}\cdot t)\cdot\mathrm{L2V}(\rho(0)))$ in a closed form. It can be obtained in polynomial time by the standard method [27].

2.3 Number Theory

▶ **Definition 1.** *A number α is* algebraic, *denoted by $\alpha\in\mathbb{A}$, if there is a nonzero $\mathbb{Q}$-polynomial $f_\alpha(z)$ of least degree, satisfying $f_\alpha(\alpha)=0$; otherwise α is* transcendental.

In the above definition, such a polynomial $f_\alpha(z)$ is called the *minimal polynomial* of α. The *degree* D of α is $\deg_z(f_\alpha)$. The standard encoding of α is the minimal polynomial f_α plus an isolation disk in the complex plane that distinguishes α from other roots of f_α.

▶ **Definition 2.** *Let $\mu_1,\ldots,\mu_m$ be irrational complex numbers. Then the* field extension *$\mathbb{Q}(\mu_1,\ldots,\mu_m):\mathbb{Q}$ is the smallest set that contains $\mu_1,\ldots,\mu_m$ and is closed under arithmetic operations, i.e. addition, subtraction, multiplication and division.*

Here those irrational complex numbers $\mu_1,\ldots,\mu_m$ are called the generators of the field extension. A field extension is *simple* if it has only one generator. For instance, the field extension $\mathbb{Q}(\sqrt{2}):\mathbb{Q}$ is exactly the set $\{a+b\sqrt{2} : a,b\in\mathbb{Q}\}$.

▶ **Lemma 3** ([32, Algorithm 2]). *Let α_1 and α_2 be two algebraic numbers of degrees D_1 and D_2, respectively. There is an algebraic number μ of degree at most $D_1 D_2$, such that the field extension $\mathbb{Q}(\mu) : \mathbb{Q}$ is exactly $\mathbb{Q}(\alpha_1, \alpha_2) : \mathbb{Q}$.*

For a collection of algebraic numbers $\alpha_1, \ldots, \alpha_m$ appearing in the input instance, by repeatedly applying this lemma, we can obtain a simple field extension $\mathbb{Q}(\mu) : \mathbb{Q}$ that can span all $\alpha_1, \ldots, \alpha_m$.

▶ **Lemma 4** ([11, Corollary 4.1.5]). *Let α be an algebraic number of degree D, and $g(z)$ an $\mathbb{A}$-polynomial with degree D_g and coefficients taken from $\mathbb{Q}(\alpha) : \mathbb{Q}$. There is a $\mathbb{Q}$-polynomial $f(z)$ of degree at most DD_g, such that roots of $g(z)$ are those of $f(z)$.*

The above lemma entails that roots of any $\mathbb{A}$-polynomial are also algebraic.

▶ **Theorem 5** (Lindemann [7, Theorem 1.4]). *For any nonzero algebraic numbers $\beta_1, \ldots, \beta_m$ and any distinct algebraic numbers $\lambda_1, \ldots, \lambda_m$, the sum $\sum_{i=1}^{m} \beta_i \mathrm{e}^{\lambda_i}$ with $m \geq 1$ is nonzero.*

▶ **Conjecture 6** (Schanuel [2]). *Let $\lambda_1, \ldots, \lambda_m$ be $\mathbb{Q}$-linearly independent complex numbers. Then the field extension $\mathbb{Q}(\lambda_1, \mathrm{e}^{\lambda_1}, \ldots, \lambda_m, \mathrm{e}^{\lambda_m}) : \mathbb{Q}$ has transcendence degree at least m.*

Let $\mathbb{A}[z_1, \ldots, z_m]$ denote the ring that contains all $\mathbb{A}$-polynomials in variables $z_1, \ldots, z_m$. Assuming Schanuel's conjecture, we could get:

▶ **Corollary 7** ([9, Proposition 5]). *Let $\lambda_1, \ldots, \lambda_m$ be $\mathbb{Q}$-linearly independent algebraic numbers. Then two co-prime elements φ_1 and φ_2 in the ring $\mathbb{A}[t, \exp(\lambda_1 t), \ldots, \exp(\lambda_m t)]$ have no common root except for 0.*

3 Quantum Continuous-Time Markov Chain

In this section, we propose the model of quantum continuous-time Markov chain (QCTMC). We will reveal that it extends the classical continuous-time Markov chain (CTMC). To show the practical usefulness, an example is further provided for modelling open quantum walk.

For the sake of clarity, we start with the QCTMC without classical states:

▶ **Definition 8.** *A* quantum continuous-time Markov chain *$\mathfrak{Q}$ is a pair $(\mathcal{H}, \mathcal{L})$, in which*

- *$\mathcal{H}$ is the Hilbert space,*
- *$\mathcal{L}$ is the transition generator function given by a Hermitian operator $\mathbf{H}$ and a finite set of linear operators $\mathbf{L}_j$ on $\mathcal{H}$.*

Usually, a density operator $\rho(0) \in \mathcal{D}$ is appointed as the initial state of $\mathfrak{Q}$.

In the model, the transition generator function $\mathcal{L}$ gives rise to a *universal* way to describe the bahavior of the QCTMC, following the generality of the Lindblad's master equation. Thus the state $\rho(t)$ is given by the closed-form solution to Eq. (5), i.e., $\mathrm{V2L}(\exp(\mathbb{M} \cdot t) \cdot \mathrm{L2V}(\rho(0)))$, where $\mathbb{M}$ is the governing matrix for $\mathcal{L}$, is a computable function from $\mathbb{R}_{\geq 0}$ to $\mathbb{C}^{d \times d}$. We notice that $0 \leq \mathrm{tr}(\mathbf{P}\rho(t)) \leq 1$ holds for any projector $\mathbf{P}$ on $\mathcal{H}$, as $\rho(t)$ is a density operator on $\mathcal{D}$. Considering computability, the entries of $\mathbf{H}$, $\mathbf{L}_j$ and $\rho(0)$ are supposed to be algebraic.

Next, we equip the QCTMC in Definition 8 with finitely many classical states.

▶ **Definition 9.** *A* quantum continuous-time Markov chain *$\mathfrak{Q}$ with a finite set S of classical states is a pair $(\mathcal{H}_{\mathrm{cq}}, \mathcal{L})$, in which*

- *$\mathcal{H}_{\mathrm{cq}} := \mathcal{C} \otimes \mathcal{H}$ is the classical–quantum system with $\mathcal{C} = \mathrm{span}(\{|s\rangle : s \in S\})$, and*
- *$\mathcal{L}$ is the transition generator function given by a Hermitian operator $\mathbf{H}$ and a finite set of linear operators $\mathbf{L}_j$ on the enlarged $\mathcal{H}_{\mathrm{cq}}$.*

Usually, a density operator $\rho(0) \in \mathcal{D}_{\mathcal{H}_{\mathrm{cq}}}$ is appointed as the initial state of $\mathfrak{Q}$.

In fact, the models in Definitions 8 and 9 have the same expressibility: The QCTMC in Definition 8 can be obtained by setting the singleton state set $S = \{s\}$ of the QCTMC in Definition 9; conversely, the QCTMC in Definition 9 can be obtained by setting the Hilbert space as $\mathcal{H}_{\mathrm{cq}}$ of the QCTMC in Definition 8. Hence, we can freely choose one of the two definitions for convenience. As an immediate result, using Definition 9, we can easily see that the QCTMC extends the CTMC by the following lemma:

▶ **Lemma 10.** *Given a CTMC $\mathfrak{C} = (S, \mathbf{Q})$, it can be modelled by a QCTMC $\mathfrak{Q} = (\mathcal{C} \otimes \mathcal{H}, \mathcal{L})$ with $\mathcal{C} = \mathrm{span}(\{|s\rangle : s \in S\})$ and $\dim(\mathcal{H}) = 1$.*

Proof. It suffices to show that the states of a CTMC $\mathfrak{C}$ can be obtained by those of some QCTMC $\mathfrak{Q}$. The state $\mathbf{x} = (x_s)_{s\in S}$ of $\mathfrak{C}$ is given by the dynamical system $\mathbf{x}'(t) = \mathbf{x}(t) \cdot \mathbf{Q}$ or equivalently its closed-form solution $\mathbf{x}(t) = \mathbf{x}(0) \cdot \exp(\mathbf{Q} \cdot t)$, where $\mathbf{x}(0)$ is a row vector interpreted as the initial state. We construct the QCTMC by setting Hermitian operator $\mathbf{H} = 0$ and linear operators $\mathbf{L}_{s,t} = |t\rangle\langle s| \otimes \mathbf{Q}[s, t]$ in Q for each pair $s, t \in S$. It is not hard to validate that the state $\rho(t)$ of $\mathfrak{Q}$ is $\mathrm{diag}(\mathbf{x}(t))$ of $\mathfrak{C}$, thus the lemma follows. ◀

▶ **Example 11** (Open Quantum Walk [35, 36])**.** Open quantum walk (OQW) is a quantum analogy of random walk, whose system evolution interacts with environment. For the sake of clarity, we suppose that a particle walking along the 2-dimensional hypercube is shown in Figure 1. The position set is $S = \{s_{00}, s_{01}, s_{10}, s_{11}\}$, where s_{00} denotes the starting position, a.k.a. the entrance, s_{01} and s_{10} denote transient positions, and s_{11} denotes the exit, an absorbing boundary. The direction set is $\{\mathrm{F}, \mathrm{S}\}$, where F means the particle takes the external transition along the first coordinate, while S means the particle takes the external transition along the second coordinate. The particle will choose a direction at every moment by the inner quantum "coin-tossing" before being absorbed. This action is implemented by the Hadamard operator $H = |+\rangle\langle \mathrm{F}| + |-\rangle\langle \mathrm{S}|$ with $|\pm\rangle = (|\mathrm{F}\rangle \pm |\mathrm{S}\rangle)/\sqrt{2}$, which denotes a fair selection between F and S as the probability amplitudes of both directions are $\frac{1}{2} = (\pm 1/\sqrt{2})^2$.

The OQW is modelled by a QCTMC $\mathfrak{Q}_1 = (\mathcal{C} \otimes \mathcal{H}, \mathcal{L})$ with $\mathcal{C} = \mathrm{span}(\{|s\rangle : s \in S\})$, where each position in S represents a classical state, and the transition function $\mathcal{L}$ is given by the Hermitian operator $\mathbf{H} = 0$ and the unique linear operator

$$\begin{aligned}\mathbf{L} =\ & |s_{10}\rangle\langle s_{00}| \otimes |\mathrm{F}\rangle\langle +| + |s_{01}\rangle\langle s_{00}| \otimes |\mathrm{S}\rangle\langle -| + |s_{11}\rangle\langle s_{01}| \otimes |\mathrm{F}\rangle\langle +|\ + \\ & |s_{00}\rangle\langle s_{01}| \otimes |\mathrm{S}\rangle\langle -| + |s_{00}\rangle\langle s_{10}| \otimes |\mathrm{F}\rangle\langle +| + |s_{11}\rangle\langle s_{10}| \otimes |\mathrm{S}\rangle\langle -| \,.\end{aligned}$$

For instance, when the particle is in the position $|s_{00}\rangle$, we first apply the quantum coin-tossing H to the state in $\mathcal{H}$, then we get the result F or S leading to the position $|s_{10}\rangle$ or $|s_{01}\rangle$. The composite operations are $(|\mathrm{F}\rangle\langle \mathrm{F}|)H = |\mathrm{F}\rangle\langle +|$ and $(|\mathrm{S}\rangle\langle \mathrm{S}|)H = |\mathrm{S}\rangle\langle -|$, which makes up the first two terms in the above $\mathbf{L}$.

From $\mathbf{L}$, we could get the governing matrix $\mathbb{M} = \mathbf{L} \otimes \mathbf{L}^* - \frac{1}{2}\mathbf{L}^\dagger\mathbf{L} \otimes \mathbf{I} - \frac{1}{2}\mathbf{I} \otimes \mathbf{L}^{\mathrm{T}}\mathbf{L}^*$. Let $\rho(0) = |s_{00}\rangle\langle s_{00}| \otimes |\mathrm{F}\rangle\langle \mathrm{F}|$ be an initial state. Then the states $\rho(t)$ of the OQW could be computed as $\mathrm{V2L}(\exp(\mathbb{M} \cdot t) \cdot \mathrm{L2V}(\rho(0)))$ (we omit the detailed value due to space limit). ⌟

As an absorbing boundary s_{11} is introduced here, two transitions from it (in dashed line) do not exist anymore. The absorbing boundary makes it impossible to characterise the system evolution by a closed system, i.e., using some Hermitian operator $\mathbf{H}$ only, as usual in [36]. Fortunately, we could characterise it by an open system, i.e., using the linear operators $\mathbf{L}_j$.

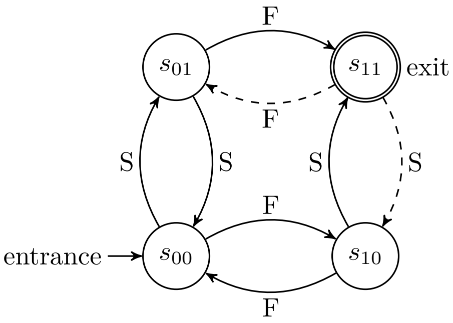


Figure 1 A sample OQW with an absorbing boundary.

4 Signal Temporal Logic

Here we recall the signal temporal logic (STL) [33], which is widely used to express real-time properties. Using it we could specify richer properties of QCTMC than linear temporal logic (LTL) and computation tree logic (CTL) in the time-bounded fragment.

▶ **Definition 12.** *The syntax of the STL formulas are defined as follows:*

$$\phi := \Phi \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \mathrm{U}^{\mathcal{I}} \phi_2$$

in which the atomic propositions Φ, interpreted as signals*, are of the form $p(\mathbf{x}) \in \mathbb{I}$ where p is a $\mathbb{Q}$-polynomial in $\mathbf{x} = (x_s)_{s\in S}$ and $\mathbb{I}$ is a rational interval, and $\mathcal{I}$ is a finite time interval. Here* U *is called the until operator, and $\phi_1 \mathrm{U}^{\mathcal{I}} \phi_2$ is the until formula.*

▶ **Definition 13.** *The semantics of the STL formulas interpreted on a QCTMC $\mathfrak{Q}$ in Definition 9 are given by the satisfaction relation $\models$:*

$$\begin{aligned}
&\rho(t) \models \Phi && \text{if } p(\mathbf{x}) \in \mathbb{I} \text{ holds with } x_s = \mathrm{tr}(\mathbf{P}_s(\rho(t))),\\
&\rho(t) \models \neg\phi && \text{if } \rho(t) \not\models \phi,\\
&\rho(t) \models \phi_1 \wedge \phi_2 && \text{if } \rho(t) \models \phi_1 \wedge \rho(t) \models \phi_2,\\
&\rho(t) \models \phi_1 \mathrm{U}^{\mathcal{I}} \phi_2 && \text{if there exists a real number } t' \in \mathcal{I},\\
& && \text{such that } \forall\, t_1 \in [t, t+t') \,:\, \rho(t_1) \models \phi_1 \text{ and } \rho(t+t') \models \phi_2,
\end{aligned}$$

where $\mathbf{P}_s$ is the projector $|s\rangle\langle s| \otimes \mathbf{I}$ onto classical state s.

From the semantics, we can see that $\Phi_1 \mathrm{U}^{\mathcal{I}_1}(\Phi_2 \mathrm{U}^{\mathcal{I}_2} \Phi_3)$ and $(\Phi_1 \mathrm{U}^{\mathcal{I}_1} \Phi_2)\mathrm{U}^{\mathcal{I}_2} \Phi_3$ have different meanings: The former formula requires there exist $t' \in \mathcal{I}_1$ and $t'' \in \mathcal{I}_2$ such that $\forall\, t_1 \in [t, t+t') : \rho(t_1) \models \Phi_1$, $\forall\, t_2 \in [t+t', t+t'+t'') : \rho(t_2) \models \Phi_2$ and $\rho(t+t'+t'') \models \Phi_3$; while the latter formula requires there exists a $t'' \in \mathcal{I}_2$ such that $\rho(t+t'') \models \Phi_3$ and for each $t_1 \in [t, t+t'')$, there exists a $t' \in \mathcal{I}_1$ such that $\forall\, t_2 \in [t_1, t_1+t') : \rho(t_2) \models \Phi_1$ and $\rho(t_1+t') \models \Phi_2$. We usually use the *parse tree* to clarify the structure of an STL formula ϕ.

The logic is very generic. STL has more expressive atomic propositions than LTL, as $\texttt{true} \equiv p(\mathbf{x}) \in (-\infty, +\infty)$ and $\texttt{false} \equiv p(\mathbf{x}) \in \emptyset$. CTL has a two-stage syntax consisting of state and path formulas. Negation and conjunction are allowed in only state formulas, not path ones. Whereas, STL allows negation and conjunction in any subformulas. Besides the standard Boolean calculus, we can easily obtain a few derivations: $\Diamond^{\mathcal{I}}\phi \equiv \texttt{true}\mathrm{U}^{\mathcal{I}}\phi$, $\Box^{\mathcal{I}}\phi \equiv \neg(\Diamond^{\mathcal{I}}\neg\phi)$, and $\phi_1 \mathrm{R}^{\mathcal{I}} \phi_2 \equiv \neg(\neg\phi_1 \mathrm{U}^{\mathcal{I}} \neg\phi_2)$ where R is the release operator.

▶ **Example 14.** Consider the open quantum walk $\mathfrak{Q}_1$ described in Example 11. It is not hard to get the probabilities of the particle staying respectively in $s_{00}, s_{01}, s_{10}, s_{11}$ as follows:

$$\begin{aligned}
x_{0,0} &= \mathrm{tr}(\mathbf{P}_{s_{0,0}}(\rho(t))) = \tfrac{1}{2}\exp(-\tfrac{2+\sqrt{2}}{2}t) + \tfrac{1}{2}\exp(-\tfrac{2-\sqrt{2}}{2}t),\\
x_{0,1} &= \mathrm{tr}(\mathbf{P}_{s_{0,1}}(\rho(t))) = -\tfrac{\sqrt{2}}{4}\exp(-\tfrac{2+\sqrt{2}}{2}t) + \tfrac{\sqrt{2}}{4}\exp(-\tfrac{2-\sqrt{2}}{2}t)\\
&\quad + \tfrac{1}{4}[\exp(-\tfrac{3}{2}t) - \exp(-\tfrac{1}{2}t)]\cos(\tfrac{1}{2}t) + \tfrac{1}{4}[\exp(-\tfrac{3}{2}t) + \exp(-\tfrac{1}{2}t)]\sin(\tfrac{1}{2}t),\\
x_{1,0} &= \mathrm{tr}(\mathbf{P}_{s_{1,0}}(\rho(t))) = -\tfrac{\sqrt{2}}{4}\exp(-\tfrac{2+\sqrt{2}}{2}t) + \tfrac{\sqrt{2}}{4}\exp(-\tfrac{2-\sqrt{2}}{2}t)\\
&\quad - \tfrac{1}{4}[\exp(-\tfrac{3}{2}t) - \exp(-\tfrac{1}{2}t)]\cos(\tfrac{1}{2}t) - \tfrac{1}{4}[\exp(-\tfrac{3}{2}t) + \exp(-\tfrac{1}{2}t)]\sin(\tfrac{1}{2}t),\\
x_{1,1} &= \mathrm{tr}(\mathbf{P}_{s_{1,1}}(\rho(t))) = 1 + \tfrac{-1+\sqrt{2}}{2}\exp(-\tfrac{2+\sqrt{2}}{2}t) - \tfrac{1+\sqrt{2}}{2}\exp(-\tfrac{2-\sqrt{2}}{2}t).
\end{aligned}$$

We can see that the probability of exiting via s_{11} would be one as t approaches infinity. A further question is asking how fast the particle would reach the exit. This is actually a question about convergence performance. It is worth monitoring the critical moment at which the probability of exiting via s_{11} equals that of staying at transient positions s_{01}, s_{10}. A requirement to be studied is like the following property.

> **Property A:** At each moment $t \in [0, 5]$, whenever the probability of staying at s_{01} or s_{10} is greater than $\frac{1}{5}$, the probability of exiting via s_{11} would exceed that of staying at s_{01} or s_{10} within the coming one unit of time.

We formally specify it with the rich STL formula

$$\phi_1 \equiv \Box^{\mathcal{I}_1}(\Phi_1 \to \Diamond^{\mathcal{I}_2}\Phi_2) \equiv \neg(\mathtt{true}\,\mathrm{U}^{\mathcal{I}_1}(\Phi_1 \wedge \neg(\mathtt{true}\,\mathrm{U}^{\mathcal{I}_2}\Phi_2))),$$

where $\mathcal{I}_1 = [0,5], \mathcal{I}_2 = [0,1]$ are time intervals and $\Phi_1 \equiv x_{0,1}+x_{1,0} > \frac{1}{5}, \Phi_2 \equiv x_{1,1} \geq x_{0,1}+x_{1,0}$ are atomic propositions, a.k.a. signals. ⌟

5 Solving Atomic Propositions

As a basic step to decide the STL formula, we need to solve the atomic proposition Φ. That is, we will compute all solutions w.r.t. t, in which $\rho(t) \models \Phi$ holds. We achieve it by a reduction to the real root isolation for a class of real-valued functions, *exponential polynomials*. Although real roots of exponential polynomials have been studied in many existing literature [1, 9, 26], the ones to be isolated in this paper involve the complicated complex exponents. So we develop a state-of-the-art real root isolation for them, whose completeness is established on Conjecture 6.

Given an atomic proposition $\Phi \equiv p(\mathbf{x}) \in \mathbb{I}$ (assuming that $\mathbb{I}$ is bounded), we would like to determine the algebraic structure of

$$\varphi(t) = (p(\mathbf{x}(t)) - \inf \mathbb{I})(p(\mathbf{x}(t)) - \sup \mathbb{I}), \tag{6}$$

with which we will design an algorithm for solving $\Phi \equiv p(\mathbf{x}) \in \mathbb{I}$. The structure of $\varphi(t)$ depends on that of $x_s(t) = \mathrm{tr}(\mathbf{P}_s(\rho(t)))$. We claim that each entry of $\rho(t)$ is of the exponential polynomial form

$$\beta_1(t)\exp(\alpha_1 t) + \beta_2(t)\exp(\alpha_2 t) + \cdots + \beta_m(t)\exp(\alpha_m t), \tag{7}$$

where $\beta_1(t), \ldots, \beta_m(t)$ are nonzero $\mathbb{A}$-polynomials and $\alpha_1, \ldots, \alpha_m$ are distinct algebraic numbers. It follows the facts:

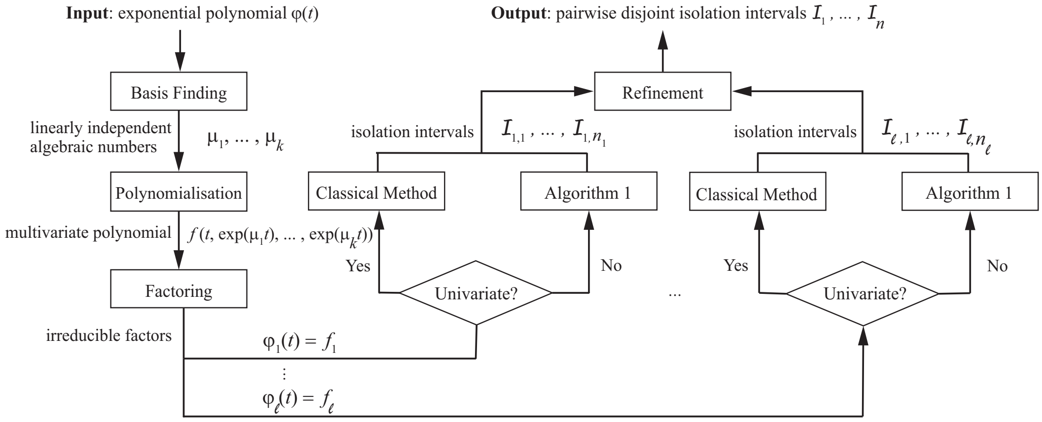

Input: exponential polynomial φ(t)
Output: pairwise disjoint isolation intervals $I_1, \dots, I_n$
Basis Finding
Refinement
linearly independent algebraic numbers
$\mu_1, \dots, \mu_k$
isolation intervals
$I_{1,1}, \dots, I_{1,n_1}$
isolation intervals
$I_{\ell,1}, \dots, I_{\ell,n_\ell}$
Polynomialisation
Classical Method
Algorithm 1
Classical Method
Algorithm 1
multivariate polynomial
$f(t, \exp(\mu_1 t), \dots, \exp(\mu_k t))$
Yes
No
Yes
No
Univariate?
...
Univariate?
Factoring
irreducible factors
$\varphi_1(t) = f_1$
⋮
$\varphi_\ell(t) = f_\ell$

Figure 2 Flow chart of the whole isolation procedure.

1. The governing matrix $\mathbb{M}$ in the state $\rho(t) = \mathrm{V2L}(\exp(\mathbb{M} \cdot t) \cdot \mathrm{L2V}(\rho(0)))$ of the QCTMC takes algebraic numbers as entries.
2. The characteristic polynomial of $\mathbb{M}$ is an $\mathbb{A}$-polynomial. The eigenvalues $\alpha_1, \dots, \alpha_m$ of $\mathbb{M}$ are algebraic, as they are roots of that $\mathbb{A}$-polynomial by Lemma 4. Those eigenvalues make up all exponents in (7).
3. The entries of the matrix exponential $\exp(\mathbb{M} \cdot t)$ are in the form (7).

The same structure holds for $x_s(t)$, as $x_s(t) = \mathrm{tr}(\mathbf{P}_s(\rho(t)))$ is simply a sum of some entries of $\rho(t)$. Furthermore, $\varphi(t)$ is also of the exponential polynomial form (7), since it is a $\mathbb{Q}$-polynomial in $\mathbf{x}(t) = (x_s(t))_{s \in S}$. If $\mathbb{I}$ is unbounded from below (resp. above), the left (resp. right) factor could be removed from (6) for further consideration.

Next, we will isolate all real roots $\lambda_1, \dots, \lambda_n$ of $\varphi(t)$ in a bounded interval $\mathcal{B}$ (to be specified in the next section). Before stating the core isolation algorithm – Algorithm 1, an overview of the isolation procedure is provided in Fig. 2. The instances to be treated can be roughly divided into two classes: one is trivial that can be solved by the classical methods, e.g., [12], for ordinary polynomials; the other is nontrivial that can be solved by Algorithm 1 but should meet three requirements of the input in Algorithm 1. After the preprocesses **Basis Finding**, **Polynomialisation** and **Factoring**, the two classes of instances can be separated and solved by the corresponding methods. In the end, the **Refinement** process outputs the pairwise disjoint isolation intervals.

The technical details along the flow chart in Fig. 2 are described below.

- **Basis Finding.** For a given set of algebraic numbers $\{\alpha_1, \dots, \alpha_m\}$ extracted from the exponents of the input exponential polynomial $\varphi(t)$, we can compute a simple extension $\mathbb{Q}(\mu) : \mathbb{Q} = \mathbb{Q}(\alpha_1, \dots, \alpha_m) : \mathbb{Q}$ by Lemma 3, such that each $\alpha_i = q_i(\mu)$ with $q_i \in \mathbb{Z}[x]$ and $\deg(q_i) < \deg(\mu)$; and further construct a $\mathbb{Q}$-linearly independent basis $\{\mu_1, \dots, \mu_k\}$ of those exponents $\{\alpha_1, \dots, \alpha_m\}$ by [26, Section 3], such that each α_i can be $\mathbb{Z}^+$-linearly expressed by $\{\mu_1, \dots, \mu_k\}$.
- **Polynomialisation.** Thus we can get a polynomial representation $f(t, \exp(\mu_1 t), \dots, \exp(\mu_k t))$ of $\varphi(t)$, where f is a $(k+1)$-variate polynomial with algebraic coefficients. That is, $\varphi(t)$ is obtained by substituting $t, \exp(\mu_1 t), \dots, \exp(\mu_k t)$ (as $k+1$ variables) into f.
- **Factoring.** Factoring $\varphi(t)$ into irreducible factors $\varphi_i(t)$ $(1 \le i \le \ell)$ corresponds to factoring the $(k+1)$-variate $\mathbb{A}$-polynomial f into irreducible factors f_i $(1 \le i \le \ell)$, which has been implemented in polynomial time, e.g. [28].

- **Univariate?** If the irreducible exponential polynomial $\varphi_i(t)$ corresponds to a univariate polynomial f_i, isolating the real roots of $\varphi_i(t)$ can be treated by classical methods [12]; otherwise we will resort to Algorithm 1, for which we can infer:
  1. The exponential polynomial $\varphi(t)$ in the form (7) is plainly an *analytic* function that is infinitely differentiable; and it is a real-valued function, or equivalently the imaginary part of $\varphi(t)$ is identically zero, since each variable $x_s(t)$ is exactly the real-valued function $\mathrm{tr}(\mathbf{P}_s\rho(t))$. The same holds for any factor $\varphi_i(t)$ of $\varphi(t)$, which ensures the first requirement of the input in Algorithm 1.
  2. By Theorem 5, thanks to the irreducibility of $\varphi_i(t)$, we have $\varphi_i(\lambda) \neq 0$ holds for any $\lambda \in \mathbb{A} \setminus \{0\}$, which ensures the second requirement of the input in Algorithm 1.
  3. By Conjucture 6 and Corollary 7, each irreducible factor $\varphi_i(t)$ and its derivative $\varphi_i'(t)$ are co-prime, and thus have no common real root except for 0, which ensures the last requirement of the input in Algorithm 1.
- **Refinement.** After performing Algorithm 1 with each individual irreducible factor $\varphi_i(t)$ of $\varphi(t)$, we would obtain a list of disjoint isolation intervals $\mathcal{I}_{i,1},\ldots,\mathcal{I}_{i,n_i}$. The isolation intervals of different irreducible factors may be overlapping. However, by Corollary 7 again, we have that each pair of co-prime factors of $\varphi(t)$ has no common real root except for 0. So all these isolation intervals $\mathcal{I}_{i,1},\ldots,\mathcal{I}_{i,n_i}$ $(1 \le i \le \ell)$ can be further refined to be pairwise disjoint. That would be the complete list of isolation intervals $\mathcal{I}_1,\ldots,\mathcal{I}_n$ for $\varphi(t)$, and thereby completes the whole isolation procedure.

Algorithm 1 Real Root Isolation for a Real-valued Function.

$$\{\mathcal{I}_1,\ldots,\mathcal{I}_n\} \Leftarrow \mathsf{Isolate}(\varphi,\mathcal{I})$$

Input: $\varphi(t)$ is a real-valued function defined on a rational interval $\mathcal{I} = [l,u]$, satisfying:
1. $\varphi(t)$ is twice-differentiable,
2. $\varphi(t)$ has no rational root in $\mathcal{I}$, and
3. $\varphi(t)$ and $\varphi'(t)$ have no common real root in $\mathcal{I}$.

Output: $\mathcal{I}_1,\ldots,\mathcal{I}_n$ are finitely many disjoint intervals, such that each contains exactly one real root of φ in $\mathcal{I}$, and together contain all.

```
compute an upper bound $M$ of $\{|\varphi'(t)| : t \in \mathcal{I}\}$;
compute an upper bound $M'$ of $\{|\varphi''(t)| : t \in \mathcal{I}\}$;
$i \leftarrow 0$, $N \leftarrow 2$ and $\delta \leftarrow (u-l)/N$;    ▷ Here $N > 1$ is a free parameter to indicate the
    number of subintervals to be split. We predefine it simply as 2.
while $i \le N$ do
    if $|\varphi(l+i\delta)| > M\delta$ then $i \leftarrow i+1$;    ▷ $\varphi$ has no local real root
    else if $|\varphi(l+i\delta+\delta)| > M\delta$ then $i \leftarrow i+2$;    ▷ $\varphi$ has no local real root
    else if $|\varphi'(l+i\delta)| \ge M'\delta$ then    ▷ $\varphi$ is locally monotonic
        if $\varphi(l+i\delta)\varphi(l+i\delta+\delta) < 0$ then output $(l+i\delta, l+i\delta+\delta)$;
        $i \leftarrow i+1$;
    else if $|\varphi'(l+i\delta+\delta)| \ge M'\delta$ then    ▷ $\varphi$ is locally monotonic
        if $\varphi(l+i\delta)\varphi(l+i\delta+2\delta) < 0$ then output $(l+i\delta, l+i\delta+2\delta)$;
        $i \leftarrow i+2$;
    else
        Isolate($\varphi$, $[l+i\delta, l+i\delta+\delta]$);
        $i \leftarrow i+1$.
```

Soundness. We justify three groups of treatment in the loop body in turn.

1. If the condition in Line 5 (resp. 6) holds, φ has no real root in the neighborhood centered at $l + i\delta$ (resp. $l + (i+1)\delta$) with radius δ. So we exclude the neighborhood.
2. If the condition in Line 7 (resp. 10) holds, φ is monotonic in the neighborhood centered at $l + i\delta$ (resp. $l + (i+1)\delta$) with radius δ. Then, if the condition in Line 8 (resp. 11) holds, i.e. φ has different signs at endpoints of the neighborhood, the unique real root in the neighborhood exists and should be output; otherwise we just exclude the neighborhood.
3. If it is not in the two decisive cases listed above, we perform Algorithm 1 recursively in a subinterval $[l + i\delta, l + (i+1)\delta]$ of $[l, u]$. ⌟

Completeness. The termination of Algorithm 1 entails the completeness. In other words, it suffices to show that for any real-valued function φ that satisfies the requirements, Algorithm 1 can always output all real roots of φ within finitely many times of recursion. Since φ and φ' are real-valued functions defined on the closed and bounded interval $\mathcal{I}$ and they have no common real root in $\mathcal{I}$, there is a positive constant τ, such that either $|\varphi| \geq \tau$ or $|\varphi'| \geq \tau$ holds everywhere of $\mathcal{I}$. Then, at any point c in $\mathcal{I}$, we can get a neighborhood with constant radius $rad := \min(\tau/M, \tau/M')$, in which either φ has no real root or is monotonic. So the two decisive cases must take place, provided that the subinterval has a length not greater than rad. It implies that the recursion depth of Algorithm 1 is bounded by $\lceil \log_2(\|\mathcal{I}\|/rad) \rceil$, where $\|\mathcal{I}\| = \sup \mathcal{I} - \inf \mathcal{I}$. Hence the termination is guaranteed. ⌟

After obtaining all real roots $\lambda_1, \ldots, \lambda_n$ of $\varphi(t)$ in the bounded interval $\mathcal{B}$ by Algorithm 1, we have that on each interval $\mathcal{J}_i$ $(0 \leq i \leq n)$ of the $n+1$ intervals in $\mathcal{B} \setminus \{\lambda_1, \ldots, \lambda_n\}$:

$$[\inf \mathcal{B}, \lambda_1), (\lambda_1, \lambda_2), \ldots, (\lambda_{n-1}, \lambda_n), (\lambda_n, \sup \mathcal{B}], \tag{8}$$

$p(\mathbf{x}(t)) \in \mathbb{I}$ holds everywhere of $t \in \mathcal{J}_i$ or nowhere. Finally, we can obtain the desired solution set $\mathbb{J}$ (to be used in the next section) of $p(\mathbf{x}(t)) \in \mathbb{I}$ by a finite union as follows

$$\bigcup_{\substack{0 \leq i \leq n \\ p(\mathbf{x}(\mathcal{J}_i)) \subseteq \mathbb{I}}} \mathcal{J}_i \cup \bigcup_{\substack{1 \leq i \leq n \\ p(\mathbf{x}(\lambda_i)) \in \mathbb{I}}} \{\lambda_i\}. \tag{9}$$

▶ **Example 15.** Reconsider Example 14. The exponential polynomial extracted from Φ_1 is

$$\varphi_1(t) = x_{0,1}(t) + x_{1,0}(t) - \tfrac{1}{5} = -\tfrac{1}{5} - \tfrac{\sqrt{2}}{2} \exp(-\tfrac{2+\sqrt{2}}{2} t) + \tfrac{\sqrt{2}}{2} \exp(-\tfrac{2-\sqrt{2}}{2} t),$$

and the exponential polynomial extracted from Φ_2 is

$$\varphi_2(t) = x_{1,1}(t) - x_{0,1}(t) - x_{1,0}(t) = 1 + (\sqrt{2} - \tfrac{1}{2}) \exp(-\tfrac{2+\sqrt{2}}{2} t) - (\sqrt{2} + \tfrac{1}{2}) \exp(-\tfrac{2-\sqrt{2}}{2} t).$$

To solve Φ_1 and Φ_2 in a bounded interval, say $\mathcal{B} = [0, 6]$, we need to determine the real roots of φ_1 and φ_2. Since both φ_1 and φ_2 are irreducible and their corresponding polynomial representations are bivariate, they have no rational root, repeated real root, nor common real root. After invoking Algorithm 1 on φ_1 with $\mathcal{B}$, we obtain two isolation intervals $[0, \frac{25}{64}]$ (containing real root $\lambda_1 \approx 0.352097$) and $[\frac{275}{64}, \frac{575}{128}]$ (containing $\lambda_2 \approx 4.49181$), which could be easily refined up to any precision. For φ_2, as $\varphi_2(0) = 0$, we make a slight shift on the left endpoint of $\mathcal{B}$ under consideration, e.g., $[\frac{1}{1000}, 6]$. After invoking Algorithm 1 on φ_2 with $[\frac{1}{1000}, 6]$, we could get the unique isolation intervals $[\frac{125059}{64000}, \frac{275117}{128000}]$, which contains the real root $\lambda_3 \approx 2.14897$. The three isolation intervals are pairwise disjoint.

Finally, we have that the solution set of Φ_1 is (λ_1, λ_2) in $\mathcal{B}$, and the solution set of Φ_2 is $\{0\} \cup [\lambda_3, 6]$. ⌟

Under Conjecture 6, we get the following computability result:

▶ **Lemma 16.** *The atomic propositions in STL are solvable over QCTMCs.*

The solvability here means that given an atomic proposition Φ and a bounded interval $\mathcal{B}$, we can always compute all solutions $\mathbb{J}$ of Φ in $\mathcal{B}$, which consists of a finite number of pairwise disjoint solution intervals $\mathcal{J}_i$ with computable real numbers as endpoints. The solving procedure is an *exact* one, i.e., no numerical error is allowed. Although it is efficient in practice, its complexity is still an open problem, as is left in existing literature [9, 26]. To pursue the guarantee of polynomial time, we could allow some numerical errors, which results in an *approximate* solving procedure.

In fact, Conjecture 6 is a powerful tool to treat roots of the general exponential polynomial. For some special subclasses of exponential polynomials, there are solid theorems to treat them: one is Theorem 5 that has been employed [1] for the exponential polynomials in the ring $\mathbb{Q}[t, \exp(t)]$, the other is the Gelfond–Schneider theorem employed [26, Subsection 4.1] for the exponential polynomials in $\mathbb{Q}[\exp(\mu_1 t), \exp(\mu_2 t)]$ where μ_1 and μ_2 are *two* $\mathbb{Q}$-linear independent *real* algebraic numbers. In Example 15, the Gelfond–Schneider theorem is sufficient to treat roots of the exponential polynomials φ_1 and φ_2, thus the termination is guaranteed. Algorithm 1 can isolate roots of elements in $\mathbb{Q}[t, \exp(\mu_1 t), \ldots, \exp(\mu_k t)]$ for *arbitrarily many* $\mathbb{Q}$-linear independent *complex* algebraic numbers $\mu_1, \ldots, \mu_k$. Additionally, Conjecture 6 has been employed to isolate simple roots of more expressible functions than exponential polynomials [38, 39], but fails to find repeated roots. Hence, this paper makes a trade-off between the expressibility of functions and the completeness of methodologies.

6 Checking STL Formulas

In the previous section, we have solved atomic propositions. Now we consider the general STL formula ϕ. For a given formula ϕ, we compute the so-called *post-monitoring period* $\mathrm{mnt}(\phi)$, independent from the initial time t_0, such that the truth of $\rho(t_0) \models \phi$ could be affected by those $\rho(t)$ with $t \in [t_0, t_0 + \mathrm{mnt}(\phi)]$. Then we decide ϕ with a bottom-up fashion. The complexity turns out to be linear in the size $\|\phi\|$ of the input STL formula ϕ, which is defined as the number of logic connectives in ϕ as standard.

Given an STL formula ϕ, we need to post-monitor a time period to decide the truth of $\rho(t_0) \models \phi$ at an initial time t_0, especially for the until formula $\phi_1 \mathrm{U}^{\mathcal{I}} \phi_2$. For example, according to the semantics of STL, to decide $\rho(t_0) \models \phi_2$ with $\phi_2 \equiv \Phi_1 \mathrm{U}^{\mathcal{I}_1}(\Phi_2 \mathrm{U}^{\mathcal{I}_2} \Phi_3)$, we have to monitor the states $\rho(t)$ from time t_0 to $t_0 + \sup \mathcal{I}_1 + \sup \mathcal{I}_2$. It inspires us to calculate the post-monitoring period $\mathrm{mnt}(\phi)$ as

$$\mathrm{mnt}(\phi) = \begin{cases} 0 & \text{if } \phi = \Phi, \\ \mathrm{mnt}(\phi_1) & \text{if } \phi = \neg\phi_1, \\ \max(\mathrm{mnt}(\phi_1), \mathrm{mnt}(\phi_2)) & \text{if } \phi = \phi_1 \wedge \phi_2, \\ \sup \mathcal{I} + \max(\mathrm{mnt}(\phi_1), \mathrm{mnt}(\phi_2)) & \text{if } \phi = \phi_1 \mathrm{U}^{\mathcal{I}} \phi_2. \end{cases} \tag{10}$$

▶ **Lemma 17.** *Given an STL formula ϕ, the satisfaction $\rho(t_0) \models \phi$ is entirely determined by the states $\rho(t)$ with $t_0 \leq t \leq t_0 + \mathrm{mnt}(\phi)$.*

Proof. We discuss it upon the syntactical structure of the STL formula ϕ.

For the atomic proposition Φ, $\rho(t_0) \models \Phi$ is plainly determined by $\rho(t_0)$.

For the negation $\neg\phi_1$, if $\rho(t_0) \models \phi_1$ is determined by $\rho(t)$ with $t_0 \leq t \leq t_0 + \mathrm{mnt}(\phi_1)$, so is $\rho(t_0) \models \neg\phi_1$.

For the conjunction $\phi_1 \wedge \phi_2$, if $\rho(t_0) \models \phi_1$ (resp. $\rho(t_0) \models \phi_2$) is determined by $\rho(t)$ with $t_0 \le t \le t_0 + \mathrm{mnt}(\phi_1)$ (resp. $t_0 \le t \le t_0 + \mathrm{mnt}(\phi_2)$), $\rho(t_0) \models \phi_1 \wedge \phi_2$ is determined by the union of those states, i.e. $\rho(t)$ with $t_0 \le t \le t_0 + \max(\mathrm{mnt}(\phi_1), \mathrm{mnt}(\phi_2))$.

For the until formula $\phi_1 \mathrm{U}^{\mathcal{I}} \phi_2$, if $\rho(t_0) \models \phi_1$ (resp. $\rho(t_0) \models \phi_2$) is determined by $\rho(t)$ with $t_0 \le t \le t_0 + \mathrm{mnt}(\phi_1)$ (resp. $t_0 \le t \le t_0 + \mathrm{mnt}(\phi_2)$), $\rho(t_0) \models \phi_1 \mathrm{U}^{\mathcal{I}} \phi_2$ is determined by $\rho(t)$ with $t_0 \le t \le t_0 + \sup \mathcal{I} + \max(\mathrm{mnt}(\phi_1), \mathrm{mnt}(\phi_2))$, where $\sup \mathcal{I}$ is caused by the admissible transition at the latest time in the until formula, since then we have to determine $\rho(t) \models \phi_1$ from time t_0 to $t_0 + \sup \mathcal{I}$ and determine $\rho(t_0 + \sup \mathcal{I}) \models \phi_2$. ◀

To decide $\rho(0) \models \phi$, our method is based on the parse tree $\mathcal{T}$ of ϕ as follows.

Basically, for each leaf of $\mathcal{T}$ that represents an atomic proposition Φ, we compute the solution set $\mathbb{J}$ (possibly a union of maximal solution intervals $\mathcal{J}$) of Φ within the monitoring interval $\mathcal{B} := [0, \mathrm{mnt}(\phi)]$ by Algorithm 1.

Inductively, for each intermediate node of $\mathcal{T}$ that represents the subformula ψ of ϕ, we tackle it into three classes.

- If $\psi = \neg\phi_1$, supposing that $\mathbb{J}_1$ is the solution set of ϕ_1, the solution set $\mathbb{J}'$ of ψ is $\mathcal{B} \setminus \mathbb{J}_1$.
- If $\psi = \phi_1 \wedge \phi_2$, supposing that $\mathbb{J}_1$ (resp. $\mathbb{J}_2$) is the solution set of ϕ_1 (resp. ϕ_2), the solution set $\mathbb{J}'$ of ψ is $\mathbb{J}_1 \cap \mathbb{J}_2$.
- If $\psi = \phi_1 \mathrm{U}^{\mathcal{I}} \phi_2$, supposing that $\mathbb{J}_1$ (resp. $\mathbb{J}_2$) is the solution set of ϕ_1 (resp. ϕ_2), the solution set $\mathbb{J}'$ of ψ is

$$\{t : (t' \in \mathcal{I}) \wedge ([t, t+t') \subseteq \mathbb{J}_1) \wedge (t + t' \in \mathbb{J}_2)\}.$$

  directly from the semantics of the until formula $\phi_1 \mathrm{U}^{\mathcal{I}} \phi_2$ in Definition 13, as
  - $[t, t+t') \subseteq \mathbb{J}_1$ if and only if $\forall\, t_1 \in [t, t+t') : \rho(t_1) \models \phi_1$, and
  - $t + t' \in \mathbb{J}_2$ if and only if $\rho(t+t') \models \phi_2$.

Note that the inductive steps of the above procedure do not generally produce all solutions of the subformula ψ in $\mathcal{B}$. Since by Lemma 17 $\rho(t_0) \models \psi$ is entirely determined by $\rho(t)$ with $t_0 \le t \le t_0 + \mathrm{mnt}(\psi)$, the resulting solution set $\mathbb{J}'$ contains all solutions of ψ in $[0, \mathrm{mnt}(\phi) - \mathrm{mnt}(\psi)]$ and possibly misses some solutions in the right subinterval $(\mathrm{mnt}(\phi) - \mathrm{mnt}(\psi), \mathrm{mnt}(\phi)]$. Anyway, the subinterval $[0, \mathrm{mnt}(\phi) - \mathrm{mnt}(\psi)]$ has the left-closed endpoint 0, which suffices to decide $\rho(0) \models \phi$.

With a bottom-up fashion, we could eventually get the solution set $\mathbb{J}$ of ϕ, by which $\rho(0) \models \phi$ can be decided to be true if and only if $0 \in \mathbb{J}$. Overall, the procedure costs $\|\phi\|$ times of the interval operations and at most $\|\phi\|$ times of calling Algorithm 1 for getting the solution set of an atomic proposition. That is, the query complexity of model checking STL formulas is linear in $\|\phi\|$ by calling Algorithm 1. The query complexity addresses the issue of the number of calls to a black box routine with unknown complexities and is commonly used in quantum computing, where the routine is usually called an oracle. For example, oracles can be a procedure of encoding an entry of a matrix into a quantum state [25] or a quantum circuit of preparing a specific quantum state [22].

▶ **Example 18.** For the STL formula $\phi_1 \equiv \neg(\mathtt{true}\, \mathrm{U}^{\mathcal{I}_1} (\Phi_1 \wedge \neg(\mathtt{true}\, \mathrm{U}^{\mathcal{I}_2} \Phi_2)))$, the post-monitoring period $\mathrm{mnt}(\phi_1)$ is $\sup \mathcal{I}_1 + \sup \mathcal{I}_2 = 6$ by Eq. (10), implying $\mathcal{B} = [0, 6]$ as used in Example 15. We construct the parse tree of ϕ_1 in Figure 3. Based on it, we could calculate the solution sets of all nodes in a bottom-up fashion:

- Basically, the solution set for Φ_1 is (λ_1, λ_2) where $\lambda_1 \approx 0.352097$ and $\lambda_2 \approx 4.49181$, the solution set for Φ_2 is $\{0\} \cup [\lambda_3, 6]$ where $\lambda_3 \approx 2.14897$, and the solution set for $\mathtt{true}$ is plainly $[0, 6]$. The post-monitoring periods of the three distinct leaves are 0. Since $\mathcal{B}$ is $[0, 6]$, we have got all the solutions in $[0, 6]$.

- The solution set for $\texttt{true}\,\mathrm{U}^{\mathcal{I}_2}\Phi_2$ is $[\lambda_3 - 1, 6]$, and that for the negation is $[0, \lambda_3 - 1)$. The post-monitoring periods of the two nodes are 1, thus we have got all solutions in $[0, 5]$.
- The solution set for $\Phi_1 \wedge \neg(\texttt{true}\,\mathrm{U}^{\mathcal{I}_2}\Phi_2)$ is $(\lambda_1, \lambda_3 - 1)$. Its post-monitoring period is 1, thus we have got all solutions in $[0, 5]$.
- Finally, the solution set for $\texttt{true}\,\mathrm{U}^{\mathcal{I}_1}(\Phi_1 \wedge \neg(\texttt{true}\,\mathrm{U}^{\mathcal{I}_2}\Phi_2))$ is $[0, \lambda_3 - 1)$, and that for the negation (the root, representing the whole STL formula ϕ_1) is $[\lambda_3 - 1, 6]$. Their post-monitoring periods are 6, thus we have got the solution in $[0, 0]$.

Since $0 \notin [\lambda_3 - 1, 6]$, we can decide $\rho(0) \models \phi_1$ to be false. Hence the particle walking along Figure 1 does not satisfy the desired convergence performance – Property A. ⌟

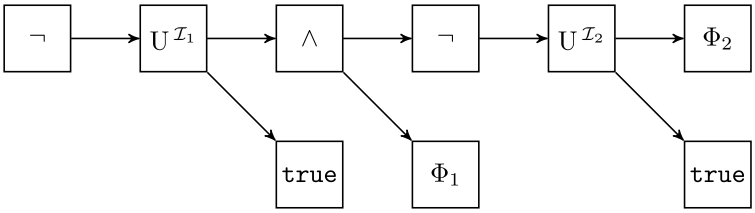


Figure 3 Parse tree of the STL formula ϕ_1

Finally, under Conjecture 6, we obtain the main result:

▶ **Theorem 19.** *The STL formulas are decidable over QCTMCs.*

7 Concluding Remarks

In this paper, we introduced the model of QCTMC that extends CTMC, and established the decidability of the STL formulas over it. To this goal, we firstly solved the atomic propositions in STL by real root isolation of a wide class of exponential polynomials, whose completeness was based on Schanuel's conjecture. Then we decided the general STL formula using interval operations with a bottom-up fashion, whose query complexity turned out to be linear in the size of the input formula by calling the developed state-of-the-art real root isolation routine. We demonstrated our method by a running example of an open quantum walk. For future work, we would like to explore the following aspects:

- how to apply the proposed method to verify non-Markov models in the real world [35];
- how to design an efficient numerical approximation of the exact method in this paper;
- and checking other formal logics, e.g. [3], over the QCTMC.

References

1 M. Achatz, S. McCallum, and V. Weispfenning. Deciding polynomial–exponential problems. In *Proc. 33rd International Symposium on Symbolic and Algebraic Computation, ISSAC 2008*, pages 215–222. ACM Press, 2008.

2 J. Ax. On Schanuel's conjectures. *Annals of Mathematics*, 93(2):252–268, 1971.

3 A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In *Computer Aided Verification: 8th International Conference, CAV'96*, volume 1102 of *LNCS*, pages 269–276. Springer, 1996.

4 A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model-checking continous-time Markov chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.

5 C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.

6 C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *CONCUR'99 Concurrency Theory*, volume 1664 of *LNCS*, pages 146–161. Springer, 1999.

7 A. Baker. *Transcendental Number Theory*. Cambridge University Press, 1975.

8 B. Baumgartner and H. Narnhofer. The structures of state space concerning quantum dynamical semigroups. *Reviews in Mathematical Physics*, 24(02):article no. 1250001, 2012.

9 V. Chonev, J. Ouaknine, and J. Worrell. On the Skolem problem for continuous linear dynamical systems. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016*, volume 55 of *LIPIcs*, pages 100:1–100:13. Schloss Dagstuhl, 2016.

10 G. I. Cirillo and F. Ticozzi. Decompositions of Hilbert spaces, stability analysis and convergence probabilities for discrete-time quantum dynamical semigroups. *Journal of Physics A: Mathematical and Theoretical*, 48(8):article no. 085302, 2015.

11 H. Cohen. *A Course in Computational Algebraic Number Theory*. Springer, 1996.

12 G. E. Collins and R. Loos. Polynomial real root isolation by differentiation. In *Proc. SYMSAC 1976*, pages 15–25. ACM Press, 1976.

13 C. Dehnert, S. Junges, J.-P. Katoen, and M. Volk. A Storm is coming: A modern probabilistic model checker. In *Computer Aided Verification - 29th International Conference, CAV 2017, Part II*, volume 10427 of *LNCS*, pages 592–600. Springer, 2017.

14 Y. Feng, E. M. Hahn, A. Turrini, and S. Ying. Model checking ω-regular properties for quantum Markov chains. In *28th International Conference on Concurrency Theory, CONCUR 2017*, volume 85 of *LIPIcs*, pages 35:1–35:16. Schloss Dagstuhl, 2017.

15 Y. Feng, N. Yu, and M. Ying. Model checking quantum Markov chains. *Journal of Computer and System Sciences*, 79(7):1181–1198, 2013.

16 S. J. Gay, R. Nagarajan, and N. Papanikolaou. Probabilistic model-checking of quantum protocols. In *Proc. 2nd International Workshop on Developments in Computational Models*, 2006. available at `arXiv:quant-ph/0504007`.

17 S. J. Gay, R. Nagarajan, and N. Papanikolaou. QMC: A model checker for quantum systems. In *Computer Aided Verification, 20th International Conference, CAV 2008*, volume 5123 of *LNCS*, pages 543–547. Springer, 2008.

18 S. Goldstein, J. L. Lebowitz, R. Tumulka, and N. Zanghì. Long-time behavior of macroscopic quantum systems. *The European Physical Journal H*, 35(2):173–200, 2010.

19 V. Gorini, A. Kossakowski, and E. C. G. Sudarshan. Completely positive dynamical semigroups of n-level systems. *Journal of Mathematical Physics*, 17(5):821–825, 1976.

20 J. Guan, Y. Feng, A. Turrini, and M. Ying. Model checking applied to quantum physics. *CoRR*, abs/1902.03218, 2019. `arXiv:1902.03218`.

21 J. Guan, Y. Feng, and M. Ying. Decomposition of quantum Markov chains and its applications. *Journal of Computer and System Sciences*, 95:55–68, 2018.

22 J. Guan, Q. Wang, and M. Ying. An HHL-based algorithm for computing hitting probabilities of quantum walks. *Quantum Information and Computation*, 2021(5&6):0395–0408, 2021.

23 J. Guan and N. Yu. Verification of continuous-time Markov chains. *CoRR*, abs/2004.08059, 2020. `arXiv:2004.08059`.

24 E. M. Hahn, Y. Li, S. Schewe, A. Turrini, and L. Zhang. iscasMc: A web-based probabilistic model checker. In *FM 2014: Formal Methods—19th International Symposium*, volume 8442 of *LNCS*, pages 312–317. Springer, 2014.

25 A. W. Harrow, A. Hassidim, and S. Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15):article no. 150502, 2009.

26 C.-C. Huang, J.-C. Li, M. Xu, and Z.-B. Li. Positive root isolation for poly-powers by exclusion and differentiation. *Journal of Symbol Computation*, 85:148–169, 2018.

27 T. Kailath. *Linear Systems*. Prentice Hall, 1980.

28 E. Kaltofen. Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization. *SIAM Journal on Computing*, 14(2):469–489, 1985.

29 M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification: 23rd International Conference, CAV 2011*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

30 L. Li and Y. Feng. Quantum Markov chains: Description of hybrid systems, decidability of equivalence, and model checking linear-time properties. *Information and Computation*, 244:229–244, 2015.

31 G. Lindblad. On the generators of quantum dynamical semigroups. *Communications in Mathematical Physics*, 48(2):119–130, 1976.

32 R. Loos. Computing in algebraic extensions. In *Computer Algebra: Symbolic and Algebraic Computation*, pages 173–187. Springer, 1983.

33 O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, volume 3253 of *LNCS*, pages 152–166. Springer, 2004.

34 M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

35 C. Pellegrini. Continuous time open quantum random walks and non-Markovian Lindblad master equations. *Journal of Statistical Physics*, 154:838–865, 2014.

36 I. Sinayskiy and F. Petruccione. Microscopic derivation of open quantum walks. *Physical Review A*, 92(3):article no. 032105, 2015.

37 W. J. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

38 A. Strzeboński. Real root isolation for exp–log functions. In *Proc. 33rd. International Symposium on Symbolic and Algebraic Computation, ISSAC 2008*, pages 303–313. ACM Press, 2008.

39 A. Strzeboński. Real root isolation for tame elementary functions. In *Proc. 34th International Symposium on Symbolic and Algebraic Computation, ISSAC 2009*, pages 341–350. ACM Press, 2009.

40 M. M. Wolf. Quantum channels & operations: Guided tour. *Lecture notes available at* `http://www-m5.ma.tum.de/foswiki/pubM`, 5, 2012.

41 M. Xu, C.-C. Huang, and Y. Feng. Measuring the constrained reachability in quantum Markov chains. *Acta Informatica, online first*, 2021.

42 M. Xu, L. Zhang, D. N. Jansen, H. Zhu, and Z. Yang. Multiphase until formulas over Markov reward models: An algebraic approach. *Theoretical Computer Science*, 611:116–135, 2016.

43 M. Ying and Y. Feng. Model-checking quantum systems. *National Science Review*, 6(1):28–31, 2019.

44 M. Ying and Y. Feng. *Model Checking Quantum Systems: Principles and Algorithms*. Cambridge University Press, 2021.

45 S. Ying, Y. Feng, N. Yu, and M. Ying. Reachability probabilities of quantum Markov chains. In *CONCUR 2013: Concurrency Theory – 24th International Conference*, volume 8052 of *LNCS*, pages 334–348. Springer, 2013.

46 N. Yu. Quantum temporal logic. *CoRR*, abs/1908.00158, 2019. `arXiv:1908.00158`.

47 N. Yu and M. Ying. Reachability and termination analysis of concurrent quantum programs. In M. Koutny and I. Ulidowski, editors, *CONCUR 2012 - Concurrency Theory - 23rd International Conference*, volume 7454 of *LNCS*, pages 69–83. Springer, 2012.

48 L. Zhang, D. N. Jansen, F. Nielson, and H. Hermanns. Automata-based CSL model checking. In *Automata, Languages and Programming: 38th International Colloquium, ICALP 2011, Part II*, volume 6756 of *LNCS*, pages 271–282. Springer, 2011.

49 L. Zhang, D. N. Jansen, F. Nielson, and H. Hermanns. Efficient CSL model checking using stratification. *Logical Methods in Computer Science*, 8(2:17):1–18, 2012.

A Unifying Framework for Deciding Synchronizability

Benedikt Bollig
Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, France

Cinzia Di Giusto
Université Côte d'Azur, CNRS, I3S, France

Alain Finkel
Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, France
Institut Universitaire de France, Paris, France

Laetitia Laversa
Université Côte d'Azur, CNRS, I3S, France

Etienne Lozes
Université Côte d'Azur, CNRS, I3S, France

Amrita Suresh
Université Paris-Saclay, ENS Paris-Saclay, CNRS, LMF, France

Abstract

Several notions of synchronizability of a message-passing system have been introduced in the literature. Roughly, a system is called synchronizable if every execution can be rescheduled so that it meets certain criteria, e.g., a channel bound. We provide a framework, based on MSO logic and (special) tree-width, that unifies existing definitions, explains their good properties, and allows one to easily derive other, more general definitions and decidability results for synchronizability.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases communicating finite-state machines, message sequence charts, synchronizability, MSO logic, special tree-width

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.14

Related Version *Full Version*: https://hal.archives-ouvertes.fr/hal-03278370

1 Introduction

Communication systems. The model of concurrent processes communicating asynchronously through FIFO channels is used since the 1960s in applications such as communication protocols [28], hardware design, MPI programs, and more recently for designing and verifying session types [23], web contracts, choreographies, concurrent programs, Erlang, Rust, etc. Since communication systems use FIFO channels, it is well known that all non-trivial properties (e.g., are all channels bounded?) are undecidable [9], essentially because a FIFO channel may simulate the tape of Turing machines and the counters of Minsky machines. However, there are many subclasses of communication systems for which the control-state reachability problem becomes decidable: e.g., synchronizable systems and existentially bounded systems (executions can be reorganized or decomposed into a finite number of sequences in which all channels are bounded), flat FIFO machines [15,17] (the graph of the machine does not contain nested loops), channel-recognizable systems [4], unreliable (lossy, insertion, duplication) FIFO systems [11], input-bounded FIFO machines [5], and half-duplex systems [10].

On the boundedness problem. We focus on the boundedness problem, which is known to be undecidable. We could limit our analysis to decide whether for a given integer $k \geq 0$, known in advance, the FIFO channels are k-bounded, and this property is generally decidable in PSPACE. Unfortunately, the k-boundedness property is too binding since we could want to design an *unbounded* system that is able, for example, to make unbounded iterations of sending and receiving messages. Hence, to cope with this limitation, one can find variants of the boundedness property that essentially reduce to say that every unbounded execution of a system (i.e., channels are unbounded along the execution) is equivalent (for instance, causally equivalent) to another *bounded* execution.

About synchronizability. To mention some examples, Lohrey and Muscholl introduced *existentially k-bounded* systems [25] (see also [18,19,24]) where all accepting executions leading to a stable (with empty channels) final configuration can be re-ordered into a k-bounded execution. This property is undecidable, even for a given k [18]. A more general definition, still called existentially bounded, is given in 2014 where the considered executions are *not* supposed to be final or stable [22]. In [21,25], the notion of *universally k-bounded* (all possible schedulings of an execution are k-bounded) is also discussed and the authors show that the property is undecidable in general. In 2011, Basu and Bultan introduced *synchronizable* systems [3], for which every execution is equivalent (for the projection on sending messages) to one of the same system but communicating by rendezvous; to avoid ambiguity, we call such systems *send-synchronizable.* In 2018, Bouajjani et al., called a system $\mathcal{S}$ *k-synchronizable* [8] (to avoid confusion we call such systems *weakly k-synchronizable*) if every MSC of $\mathcal{S}$ admits a linearization (which is not necessarily an execution) that can be divided into blocks of at most k messages. After each block, a message is either read, or will never be read. This constraint seems to imply that buffers are bounded to k messages. However, as the linearization need not be an execution, this implies that a weakly k-synchronizable execution, even with the more efficient reschedule, can need unbounded channels to be run by the system.

Communication architecture and variants. A key difference between these works is that they consider different communication architectures. Existentially bounded systems have been studied for p2p (with one queue per pair of processes), whereas k-synchronizability has been studied for mailbox communication, for which each process merges all its incoming messages in a unique queue. The decidability results for k-synchronizability have been extended to p2p communications [14], but it is unknown whether the decidability results for existentially bounded systems extend to mailbox communication. Moreover, variants of those definitions can be obtained depending on if we consider messages that are sent but never read, called unmatched messages. Indeed the challenges that arise in [8] are due to mailbox communication and unmatched messages blocking a channel so that all messages sent afterwards will never be read. To clarify and overcome this issue, we propose *strong k-synchronizability*, a new definition that is suitable for mailbox communication: an execution is called *strongly k-synchronizable* if it can be rescheduled into another k-bounded execution such that there are at most k messages in the channels before emptying them.

Contributions. Our contributions can be summarized as follows:

- In order to unify the notions of synchronizability, we introduce a general framework based on monadic second-order (MSO) logic and (special) tree-width that captures most existing definitions of systems that may work with bounded channels. Moreover, reachability and model checking are shown decidable in this framework.

- We show that existentially bounded systems can be expressed in our framework and, as a consequence, the existentially k-bounded property is decidable by using the generic proof.
- We generalize the existing notion of (weak) k-synchronizability in [8] and we introduce three new classes of synchronizable systems: weakly synchronizable (which are more general than weakly k-synchronizable), strongly synchronizable and strongly k-synchronizable (which are particular cases of weakly synchronizable). We then prove that these properties all fit in our framework and are all shown decidable using the generic proof.
- We then deduce that reachability and model checking are decidable for these classes (only control-state reachability was shown to be decidable for weakly k-synchronizable in [8] and it is clearly also decidable for existentially/universally bounded systems but reachability properties are generally not studied for these classes of systems).
- In order to obtain better complexity results for some classes (strongly and weakly synchronizable systems), we also use the fragment of propositional dynamic logic with loop and converse (LCPDL) instead of MSO logic in our framework.
- We provide a comparison between synchronizable classes both for p2p and mailbox semantics (see Fig. 8 for p2p systems and Fig. 9 for mailbox systems). In particular, we clarify the link between weakly synchronizable and existentially bounded systems for both p2p and mailbox systems, which was left open in [8] and solved only for p2p systems in [23, Theorem 7] where weakly synchronizable systems are shown to be included into existentially bounded ones when considering executions (and not MSCs as in our case).

Outline. Section 2 defines some preliminary notions such as p2p/mailbox message sequence charts (MSCs), and communicating systems. Section 3 presents the unifying MSO framework and two general theorems on k-synchronizability and model checking. In Section 4, we apply the MSO framework to different existing definitions of synchronizability, and we introduce a new decidable one. Section 5 studies the relations between the classes. In Section 6, we conclude with some final remarks. Due to space constraints, some proofs are given in the full version of the paper, available at: `https://hal.archives-ouvertes.fr/hal-03278370`

2 Preliminaries

2.1 Message Sequence Charts

Assume a finite set of processes $\mathbb{P}$ and a finite set of messages $\mathbb{M}$. The set of (p2p) channels is $\mathbb{C} = \{(p,q) \in \mathbb{P} \times \mathbb{P} \mid p \neq q\}$. A send action is of the form $\mathit{send}(p,q,m)$ where $(p,q) \in \mathbb{C}$ and $m \in \mathbb{M}$. It is executed by p and sends message m to q. The corresponding receive action, executed by q, is $\mathit{rec}(p,q,m)$. For $(p,q) \in \mathbb{C}$, let $\mathit{Send}(p,q,\_) = \{\mathit{send}(p,q,m) \mid m \in \mathbb{M}\}$ and $\mathit{Rec}(p,q,\_) = \{\mathit{rec}(p,q,m) \mid m \in \mathbb{M}\}$. For $p \in \mathbb{P}$, we set $\mathit{Send}(p,\_,\_) = \{\mathit{send}(p,q,m) \mid q \in \mathbb{P} \setminus \{p\}$ and $m \in \mathbb{M}\}$, etc. Moreover, $\Sigma_p = \mathit{Send}(p,\_,\_) \cup \mathit{Rec}(\_,p,\_)$ will denote the set of all actions that are executed by p. Finally, $\Sigma = \bigcup_{p \in \mathbb{P}} \Sigma_p$ is the set of all the actions.

Peer-to-peer MSCs. A *p2p MSC* (or simply *MSC*) over $\mathbb{P}$ and $\mathbb{M}$ is a tuple $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ where $\mathcal{E}$ is a finite (possibly empty) set of *events* and $\lambda : \mathcal{E} \to \Sigma$ is a labeling function. For $p \in \mathbb{P}$, let $\mathcal{E}_p = \{e \in \mathcal{E} \mid \lambda(e) \in \Sigma_p\}$ be the set of events that are executed by p. We require that $\rightarrow$ (the *process relation*) is the disjoint union $\bigcup_{p \in \mathbb{P}} \rightarrow_p$ of relations $\rightarrow_p \subseteq \mathcal{E}_p \times \mathcal{E}_p$ such that $\rightarrow_p$ is the direct successor relation of a total order on $\mathcal{E}_p$. For an event $e \in \mathcal{E}$, a set of actions $A \subseteq \Sigma$, and a relation $R \subseteq \mathcal{E} \times \mathcal{E}$, let $\#_A(R,e) = |\{f \in \mathcal{E} \mid (f,e) \in R$ and $\lambda(f) \in A\}|$. We require that $\lhd \subseteq \mathcal{E} \times \mathcal{E}$ (the *message relation*) satisfies the following:

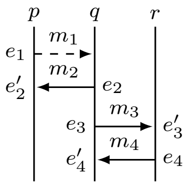


Figure 1 MSC M_1.

(1) for every pair $(e, f) \in \lhd$, there is a send action $send(p, q, m) \in \Sigma$ such that $\lambda(e) = send(p, q, m)$, $\lambda(f) = rec(p, q, m)$, and $\#_{Send(p,q,\_)}(\rightarrow^+, e) = \#_{Rec(p,q,\_)}(\rightarrow^+, f)$,
(2) for all $f \in \mathcal{E}$ such that $\lambda(f)$ is a receive action, there is $e \in \mathcal{E}$ such that $e \lhd f$.

Finally, letting $\leq_M = (\rightarrow \cup \lhd)^*$, we require that $\leq_M$ is a partial order.

Condition (1) above ensures that every (p2p) channel (p, q) behaves in a FIFO manner. By Condition (2), every receive event has a matching send event. Note that, however, there may be unmatched send events in an MSC. We let $SendEv(M) = \{e \in \mathcal{E} \mid \lambda(e)$ is a send action$\}$, $RecEv(M) = \{e \in \mathcal{E} \mid \lambda(e)$ is a receive action$\}$, $Matched(M) = \{e \in \mathcal{E} \mid$ there is $f \in \mathcal{E}$ such that $e \lhd f\}$, and $Unm(M) = \{e \in \mathcal{E} \mid \lambda(e)$ is a send action and there is no $f \in \mathcal{E}$ such that $e \lhd f\}$. We do not distinguish isomorphic MSCs and let MSC be the set of all MSCs over the given sets $\mathbb{P}$ and $\mathbb{M}$.

▶ **Example 1.** For a set of processes $\mathbb{P} = \{p, q, r\}$ and a set of messages $\mathbb{M} = \{m_1, m_2, m_3, m_4\}$, $M_1 = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ is an MSC where, for example, $e_2 \lhd e'_2$ and $e'_3 \rightarrow e_4$. The dashed arrow means that the send event e_1 does not have a matching receive, so $e_1 \in Unm(M_1)$. Moreover, $e_2 \leq_{M_1} e_4$, but $e_1 \not\leq_{M_1} e_4$. We can find a total order $\rightsquigarrow \supseteq \leq_{M_1}$ such that $e_1 \rightsquigarrow e_2 \rightsquigarrow e'_2 \rightsquigarrow e_3 \rightsquigarrow e'_3 \rightsquigarrow e_4 \rightsquigarrow e'_4$. We call $\rightsquigarrow$ a linearization, which is formally defined below.

Mailbox MSCs. For an MSC $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$, we define an additional binary relation that represents a constraint under the mailbox semantics, where each process has only one incoming channel. Let $\sqsubset_M \subseteq \mathcal{E} \times \mathcal{E}$ be defined by: $e_1 \sqsubset_M e_2$ if there is $q \in \mathbb{P}$ such that $\lambda(e_1) \in Send(\_, q, \_)$, $\lambda(e_2) \in Send(\_, q, \_)$, and one of the following holds:

- $e_1 \in Matched(M)$ and $e_2 \in Unm(M)$, or
- $e_1 \lhd f_1$ and $e_2 \lhd f_2$ for some $f_1, f_2 \in \mathcal{E}_q$ such that $f_1 \rightarrow^+ f_2$.

We let $\preceq_M = (\rightarrow \cup \lhd \cup \sqsubset_M)^*$. Note that $\leq_M \subseteq \preceq_M$. We call $M \in \mathsf{MSC}$ a *mailbox MSC* if $\preceq_M$ is a partial order. Intuitively, this means that events can be scheduled in a way that corresponds to the mailbox semantics, i.e., with one incoming channel per process. Following the terminology in [8], we also say that a mailbox MSC satisfies *causal delivery*. The set of mailbox MSCs $M \in \mathsf{MSC}$ is denoted by $\mathsf{MSC}_{\mathsf{mb}}$.

▶ **Example 2.** MSC M_1 is a mailbox MSC. Indeed, even though the order $\rightsquigarrow$ defined in Example 1 does not respect all mailbox constraints, particularly the fact that $e_4 \sqsubset_{M_1} e_1$, there is a total order $\rightsquigarrow \supseteq \preceq_{M_1}$ such that $e_2 \rightsquigarrow e_3 \rightsquigarrow e'_3 \rightsquigarrow e_4 \rightsquigarrow e_1 \rightsquigarrow e'_2 \rightsquigarrow e'_4$. We call $\rightsquigarrow$ a mailbox linearization, which is formally defined below.

Linearizations, Prefixes, and Concatenation. Consider $M = (\mathcal{E}, \rightarrow, \lhd, \lambda) \in \mathsf{MSC}$. A *p2p linearization* (or simply *linearization*) of M is a (reflexive) total order $\rightsquigarrow \subseteq \mathcal{E} \times \mathcal{E}$ such that $\leq_M \subseteq \rightsquigarrow$. Similarly, a *mailbox linearization* of M is a total order $\rightsquigarrow \subseteq \mathcal{E} \times \mathcal{E}$ such that

$\preceq_M \subseteq \rightsquigarrow$. That is, every mailbox linearization is a p2p linearization, but the converse is not necessarily true (Example 2). Note that an MSC is a mailbox MSC iff it has at least one mailbox linearization.

Let $M = (\mathcal{E}, \rightarrow, \lhd, \lambda) \in \mathsf{MSC}$ and consider $E \subseteq \mathcal{E}$ such that E is $\leq_M$*-downward-closed*, i.e, for all $(e, f) \in \leq_M$ such that $f \in E$, we also have $e \in E$. Then, the MSC $(E, \rightarrow \cap (E \times E), \lhd \cap (E \times E), \lambda')$, where λ' is the restriction of $\mathcal{E}$ to E, is called a *prefix* of M. In particular, the empty MSC is a prefix of M. We denote the set of prefixes of M by $\mathit{Pref}(M)$. This is extended to sets $L \subseteq \mathsf{MSC}$ as expected, letting $\mathit{Pref}(L) = \bigcup_{M \in L} \mathit{Pref}(M)$.

▶ **Lemma 3.** *Every prefix of a mailbox MSC is a mailbox MSC.*

Let $M_1 = (\mathcal{E}_1, \rightarrow_1, \lhd_1, \lambda_1)$ and $M_2 = (\mathcal{E}_2, \rightarrow_2, \lhd_2, \lambda_2)$ be two MSCs. Their *concatenation* $M_1 \cdot M_2 = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ is defined if, for all $(p, q) \in \mathbb{C}$, $e_1 \in \mathit{Unm}(M_1)$, and $e_2 \in \mathcal{E}_2$ such that $\lambda(e_1) \in \mathit{Send}(p, q, \_)$ and $\lambda(e_2) \in \mathit{Send}(p, q, \_)$, we have $e_2 \in \mathit{Unm}(M_2)$. As expected, $\mathcal{E}$ is the disjoint union of $\mathcal{E}_1$ and $\mathcal{E}_2$, $\lhd = \lhd_1 \cup \lhd_2$, λ is the "union" of λ_1 and λ_2, and $\rightarrow = \rightarrow_1 \cup \rightarrow_2 \cup R$. Here, R contains, for all $p \in \mathbb{P}$ such that $(\mathcal{E}_1)_p$ and $(\mathcal{E}_2)_p$ are non-empty, the pair (e_1, e_2) where e_1 is the maximal p-event in M_1 and e_2 is the minimal p-event in M_2. Note that $M_1 \cdot M_2$ is indeed an MSC and that concatenation is associative.

2.2 Communicating Systems

We now recall the definition of communicating systems (aka communicating finite-state machines or message-passing automata), which consist of finite-state machines A_p (one for every process $p \in \mathbb{P}$) that can communicate through the FIFO channels from $\mathbb{C}$.

▶ **Definition 4.** *A* communicating system *over $\mathbb{P}$ and $\mathbb{M}$ is a tuple $\mathcal{S} = (A_p)_{p \in \mathbb{P}}$. For each $p \in \mathbb{P}$, $A_p = (\mathit{Loc}_p, \delta_p, \ell_p^0)$ is a finite transition system where Loc_p is a finite set of local (control) states, $\delta_p \subseteq \mathit{Loc}_p \times \Sigma_p \times \mathit{Loc}_p$ is the transition relation, and $\ell_p^0 \in \mathit{Loc}_p$ is the initial state.*

Given $p \in \mathbb{P}$ and a transition $t = (\ell, a, \ell') \in \delta_p$, we let $\mathit{source}(t) = \ell$, $\mathit{target}(t) = \ell'$, $\mathit{action}(t) = a$, and $\mathit{msg}(t) = m$ if $a \in \mathit{Send}(\_, \_, m) \cup \mathit{Rec}(\_, \_, m)$.

There are in general two ways to define the semantics of a communicating system. Most often it is defined as a global infinite transition system that keeps track of the various local control states and all (unbounded) channel contents. As, in this paper, our arguments are based on a graph view of MSCs, we will define the language of $\mathcal{S}$ directly as a set of MSCs. These two semantic views are essentially equivalent, but they have different advantages depending on the context. We refer to [1] for a thorough discussion.

Let $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ be an MSC. A *run* of $\mathcal{S}$ on M is a mapping $\rho : \mathcal{E} \rightarrow \bigcup_{p \in \mathbb{P}} \delta_p$ that assigns to every event e the transition $\rho(e)$ that is executed at e. Thus, we require that

(i) for all $e \in \mathcal{E}$, we have $\mathit{action}(\rho(e)) = \lambda(e)$,
(ii) for all $(e, f) \in \rightarrow$, $\mathit{target}(\rho(e)) = \mathit{source}(\rho(f))$,
(iii) for all $(e, f) \in \lhd$, $\mathit{msg}(\rho(e)) = \mathit{msg}(\rho(f))$, and
(iv) for all $p \in \mathbb{P}$ and $e \in \mathcal{E}_p$ such that there is no $f \in \mathcal{E}$ with $f \rightarrow e$, we have $\mathit{source}(\rho(e)) = \ell_p^0$.

Letting run $\mathcal{S}$ directly on MSCs is actually very convenient. This allows us to associate with $\mathcal{S}$ its p2p language and mailbox language in one go. The *p2p language* of $\mathcal{S}$ is $L_{\mathsf{p2p}}(\mathcal{S}) = \{M \in \mathsf{MSC} \mid \text{there is a run of } \mathcal{S} \text{ on } M\}$. The *mailbox language* of $\mathcal{S}$ is $L_{\mathsf{mb}}(\mathcal{S}) = \{M \in \mathsf{MSC}_{\mathsf{mb}} \mid \text{there is a run of } \mathcal{S} \text{ on } M\}$.

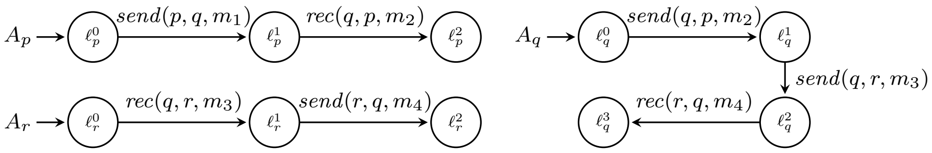


Figure 2 System $\mathcal{S}_1$.

Note that, following [8, 14], we do not consider final states or final configurations, as our purpose is to reason about all possible traces that can be *generated* by $\mathcal{S}$. The next lemma is obvious for the p2p semantics and follows from Lemma 3 for the mailbox semantics.

▶ **Lemma 5.** *For all* com $\in \{\mathsf{p2p}, \mathsf{mb}\}$*, $L_{\text{com}}(\mathcal{S})$ is prefix-closed: $Pref(L_{\text{com}}(\mathcal{S})) \subseteq L_{\text{com}}(\mathcal{S})$.*

▶ **Example 6.** Fig. 2 depicts $\mathcal{S}_1 = (A_p, A_q, A_r)$ such that MSC M_1 in Fig. 1 belongs to $L_{\mathsf{p2p}}(\mathcal{S}_1)$ and to $L_{\mathsf{mb}}(\mathcal{S}_1)$. There is a unique run ρ of $\mathcal{S}_1$ on M_1. We can see that $(e_3', e_4) \in \rightarrow$ and $target(\rho(e_3')) = source(\rho(e_4)) = \ell_r^1$, $(e_2, e_2') \in \lhd_{M_1}$, and $msg(\rho(e_2)) = msg(\rho(e_2')) = m_2$.

2.3 Conflict Graph

We now recall the notion of a conflict graph associated to an MSC defined in [8]. This graph is used to depict the causal dependencies between message exchanges. Intuitively, we have a dependency whenever two messages have a process in common. For instance, an $\xrightarrow{SS}$ dependency between message exchanges v and v' expresses the fact that v' has been sent after v, by the same process. This notion is of interest because it was seen in [8] that the notion of synchronizability in MSCs (which is studied in this paper) can be graphically characterized by the nature of the associated conflict graph. It is defined in terms of linearizations in [14], but we equivalently express it directly in terms of MSCs.

For an MSC $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ and $e \in \mathcal{E}$, we define the type $\tau(e) \in \{S, R\}$ of e by $\tau(e) = S$ if $e \in SendEv(M)$ and $\tau(e) = R$ if $e \in RecEv(M)$. Moreover, for $e \in Unm(M)$, we let $\mu(e) = e$, and for $(e, e') \in \lhd$, we let $\mu(e) = \mu(e') = (e, e')$.

▶ **Definition 7** (Conflict graph)**.** *The* conflict graph $\mathsf{CG}(M)$ *of an MSC $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ is the labeled graph $(Nodes, Edges)$, with $Edges \subseteq Nodes \times \{S, R\}^2 \times Nodes$, defined by $Nodes = \lhd \cup Unm(M)$ and $Edges = \{(\mu(e), \tau(e)\tau(f), \mu(f)) \mid (e, f) \in \rightarrow^+\}$. In particular, a node of* $\mathsf{CG}(M)$ *is either a single unmatched send event or a message pair $(e, e') \in \lhd$.*

3 Model Checking and Synchronizability

In this section, we survey two classical decision problems for communicating systems. The first problem is the model-checking problem, in which one checks whether a given system satisfies a given specification. A canonical specification language for MSCs is monadic second-order (MSO) logic. However, model checking in full generality is undecidable. A common approach is, therefore, to restrict the behavior of the given system to MSCs of bounded (special) tree-width. Next, we introduce MSO logic and special tree-width.

3.1 Logic and Special Tree-Width

Monadic Second-Order Logic. The set of MSO formulas over MSCs (over $\mathbb{P}$ and $\mathbb{M}$) is given by the grammar $\varphi ::= x \rightarrow y \mid x \lhd y \mid \lambda(x) = a \mid x = y \mid x \in X \mid \exists x.\varphi \mid \exists X.\varphi \mid \varphi \vee \varphi \mid \neg\varphi$, where $a \in \Sigma$, x and y are first-order variables, interpreted as events of an MSC, and X is a

$$\begin{array}{lll}
M \models \mathsf{E}\sigma \ \text{ if } \ [\![\sigma]\!]_M \neq \emptyset & & [\![\rightarrow]\!]_M := \rightarrow \quad \text{and} \quad [\![\lhd]\!]_M := \lhd \\
[\![a]\!]_M & := \{e \in \mathcal{E} \mid \lambda(e) = a\} & [\![\mathsf{test}(\sigma)]\!]_M := \{(e,e) \mid e \in [\![\sigma]\!]_M\} \\
[\![\langle\pi\rangle\sigma]\!]_M & := \{e \in \mathcal{E} \mid \exists f \in [\![\sigma]\!]_M : (e,f) \in [\![\pi]\!]_M\} & [\![\mathsf{jump}]\!]_M := \mathcal{E} \times \mathcal{E} \\
[\![\mathsf{Loop}\langle\pi\rangle]\!]_M & := \{e \in \mathcal{E} \mid (e,e) \in [\![\pi]\!]_M\} & [\![\pi_1 + \pi_2]\!]_M := [\![\pi_1]\!]_M \cup [\![\pi_2]\!]_M \\
[\![\pi^{-1}]\!]_M & := \{(e,f) \in \mathcal{E} \times \mathcal{E} \mid (f,e) \in [\![\pi]\!]_M\} & [\![\pi^*]\!]_M := \bigcup_{n \in \mathbb{N}} [\![\pi]\!]_M^n \\
[\![\pi_1 \cdot \pi_2]\!]_M & := \{(e,f) \in \mathcal{E} \times \mathcal{E} \mid \exists g \in \mathcal{E} : (e,g) \in [\![\pi_1]\!]_M \text{ and } (g,f) \in [\![\pi_2]\!]_M\} &
\end{array}$$

Figure 3 Semantics of LCPDL.

second-order variable, interpreted as a set of events. We assume that we have an infinite supply of variables, and we use common abbreviations such as $\wedge$, $\forall$, etc. The satisfaction relation is defined in the standard way and self-explanatory. For example, the formula $\neg\exists x.(\bigvee_{a \in Send(\_,\_,\_)} \lambda(x) = a \ \wedge \ \neg matched(x))$ with $matched(x) = \exists y. x \lhd y$ says that there are no unmatched send events. It is not satisfied by MSC M_1 of Fig. 1, as message m_1 is not received, but by M_4 from Fig. 6.

Given a sentence φ, i.e., a formula without free variables, we let $L(\varphi)$ denote the set of (p2p) MSCs that satisfy φ. It is worth mentioning that the (reflexive) transitive closure of a binary relation defined by an MSO formula with free variables x and y, such as $x \rightarrow y$, is MSO-definable so that the logic can freely use formulas of the form $x \rightarrow^+ y$ or $x \leq y$ (where $\leq$ is interpreted as $\leq_M$ for the given MSC M). Therefore, the definition of a mailbox MSC can be readily translated into the formula $\varphi_{\mathsf{mb}} = \neg\exists x.\exists y.(\neg(x = y) \wedge x \preceq y \wedge y \preceq x)$ so that we have $L(\varphi_{\mathsf{mb}}) = \mathsf{MSC}_{\mathsf{mb}}$. Here, $x \preceq y$ is obtained as the MSO-definable reflexive transitive closure of the union of the MSO-definable relations $\rightarrow$, $\lhd$, and $\sqsubset$. In particular, we may define $x \sqsubset y$ by:

$$x \sqsubset y = \bigvee_{\substack{q \in \mathbb{P} \\ a,b \in Send(\_,q,\_)}} \lambda(x) = a \ \wedge \ \lambda(y) = b \wedge \left(\begin{array}{ll} & matched(x) \wedge \neg matched(y) \\ \vee & \exists x'.\exists y'.(x \lhd x' \ \wedge \ y \lhd y' \ \wedge \ x' \rightarrow^+ y') \end{array} \right).$$

Propositional Dynamic Logic (PDL). For better complexity, we also consider PDL with Loop and Converse, henceforth called LCPDL (cf. [6, 7, 27] for more details). Its syntax is:

$$\begin{array}{ll}
\Phi ::= \mathsf{E}\sigma \mid \Phi \vee \Phi \mid \neg\Phi & \text{(sentence)} \\
\sigma ::= a \mid \sigma \vee \sigma \mid \neg\sigma \mid \langle\pi\rangle\sigma \mid \mathsf{Loop}\langle\pi\rangle & \text{(event formula)} \\
\pi ::= \rightarrow \mid \lhd \mid \mathsf{test}(\sigma) \mid \mathsf{jump} \mid \pi + \pi \mid \pi \cdot \pi \mid \pi^* \mid \pi^{-1} & \text{(path formula)}
\end{array}$$

where $a \in \Sigma$. We use the symbol $\top$ to denote a tautology event formula (such as $a \vee \neg a$). We describe the semantics for the logic in Fig. 3 (apart from the obvious cases). A sentence Φ is evaluated wrt. an MSC $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$. An event formula σ is evaluated wrt. M and an event $e \in \mathcal{E}$ so that it defines a unary relation $[\![\sigma]\!]_M \subseteq \mathcal{E}$. Finally, a path formula π is evaluated over two events, and so it defines a binary relation $[\![\pi]\!]_M \subseteq \mathcal{E} \times \mathcal{E}$. Finally, we let $L(\Phi) = \{M \in \mathsf{MSC} \mid M \models \Phi\}$. Note that every LCPDL-definable property is MSO-definable.

It can be seen below that the mailbox semantics can be readily translated into the LCPDL formula $\Phi_{\mathsf{mb}} = \neg\mathsf{E}\,(\mathsf{Loop}\langle(\lhd + \rightarrow + \sqsubset)^+\rangle)$ such that $L(\Phi_{\mathsf{mb}}) = \mathsf{MSC}_{\mathsf{mb}}$. Hereby, we let

$$\sqsubset \, = \lhd \cdot \rightarrow^+ \cdot \lhd^{-1} \ + \sum_{\substack{q \in \mathbb{P} \\ a,b \in Send(\_,q,\_)}} \mathsf{test}(a) \cdot \lhd \cdot \mathsf{jump} \cdot \mathsf{test}(b \wedge \neg\langle\lhd\rangle\top)\,.$$

Special Tree-Width. *Special tree-width* [12], is a graph measure that indicates how close a graph is to a tree (we may also use classical *tree-width* instead). This or similar measures are commonly employed in verification. For instance, tree-width and split-width have been used in [26] and, respectively, [2, 13] to reason about graph behaviors generated by pushdown and queue systems. There are several ways to define the special tree-width of an MSC. We adopt the following game-based definition from [7].

Adam and Eve play a two-player turn based "decomposition game" whose positions are MSCs with some pebbles placed on some events. More precisely, Eve's positions are *marked MSC fragments* (M, U), where $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ is an *MSC fragment* (an MSC with possibly some edges from $\lhd$ or $\rightarrow$ removed) and $U \subseteq \mathcal{E}$ is the subset of marked events. Adam's positions are pairs of marked MSC fragments. A move by Eve consists in the following steps:

1. marking some events of the MSC resulting in (M, U') with $U \subseteq U' \subseteq \mathcal{E}$,
2. removing (process and/or message) edges whose endpoints are marked,
3. dividing (M, U) in (M_1, U_1) and (M_2, U_2) such that M is the disjoint (unconnected) union of M_1 and M_2 and marked nodes are inherited.

When it is Adam's turn, he simply chooses one of the two marked MSC fragments. The initial position is $(M, \emptyset)$ where M is the (complete) MSC at hand. A terminal position is any position belonging to Eve such that all events are marked. For $k \in \mathbb{N}$, we say that the game is k-winning for Eve if she has a (positional) strategy that allows her, starting in the initial position and independently of Adam's moves, to reach a terminal position such that, in every single position visited along the play, there are at most $k + 1$ marked events.

▶ **Fact 8** ([7])**.** *The special tree-width of an MSC is the least k such that the associated game is k-winning for Eve.*

The set of MSCs whose special tree-width is at most k is denoted by $\mathsf{MSC}^{k\text{-}\mathsf{stw}}$.

3.2 Model Checking

In general, even simple verification problems, such as control-state reachability, are undecidable for communicating systems [9]. However, they are decidable when we restrict to behaviors of bounded special tree-width, which motivates the following definition of a generic **bounded model-checking problem** for com $\in \{\mathsf{p2p}, \mathsf{mb}\}$:

Input: Two finite sets $\mathbb{P}$ and $\mathbb{M}$, a communicating system $\mathcal{S}$, an MSO sentence φ, and $k \in \mathbb{N}$ (given in unary).

Question: Do we have $L_{\mathrm{com}}(\mathcal{S}) \cap \mathsf{MSC}^{k\text{-}\mathsf{stw}} \subseteq L(\varphi)$?

▶ **Fact 9** ([7])**.** *The bounded model-checking problem for* com $= \mathsf{p2p}$ *is decidable. When the formulas φ are from LCPDL, then the problem is solvable in exponential time.*

Note that [7] does not employ the LCPDL modality jump, but it can be integrated easily. Using φ_{mb} or Φ_{mb}, we obtain the corresponding result for mailbox systems as a corollary:

▶ **Theorem 10.** *The bounded model-checking problem for* com $= \mathsf{mb}$ *is decidable. When the formulas φ are from LCPDL, then the problem is solvable in exponential time.*

3.3 Synchronizability

The above model-checking approach is incomplete in the sense that a positive answer does not imply correctness of the whole system. The system may still produce behaviors of special tree-width greater than k that violate the given property. However, if we know that a system only generates behaviors from a class whose special tree-width is bounded by k, we can still conclude that the system is correct.

Table 1 Summary of the decidability of the synchronizability problem in various classes.

	PEER-TO-PEER	MAILBOX
Weakly synchronous	Undecidable [Thm. 22]	EXPTIME [Thm. 21]
Weakly k-synchronous	Decidable [8, 14] and [Thm. 29]	
Strongly k-synchronous	—	Decidable [Thm. 35]
Existentially k-p2p-bounded	Decidable [18, Prop. 5.5]	
Existentially k-mailbox-bounded	—	Decidable [Prop. 40]

This motivates the *synchronizability problem*. Several notions of synchronizability have been introduced in the literature. However, they all amount to asking whether all behaviors generated by a given communicating system have a particular shape, i.e., whether they are all included in a fixed (or given) set of MSCs $\mathcal{C}$. Thus, the synchronizability problem is essentially an inclusion problem, namely $L_{\mathsf{p2p}}(\mathcal{S}) \subseteq \mathcal{C}$ or $L_{\mathsf{mb}}(\mathcal{S}) \subseteq \mathcal{C}$. We show that, for decidability, it is enough to have that $\mathcal{C}$ is MSO-definable and special-tree-width-bounded (STW-bounded): We call $\mathcal{C} \subseteq \mathsf{MSC}$

(i) *MSO-definable* if there is an MSO-formula φ such that $L(\varphi) = \mathcal{C}$,
(ii) *LCPDL-definable* if there is an an LCPDL-formula Φ such that $L(\Phi) = \mathcal{C}$,
(iii) *STW-bounded* if there is $k \in \mathbb{N}$ such that $\mathcal{C} \subseteq \mathsf{MSC}^{k\text{-stw}}$.

An important component of the decidability proof is the following lemma, which shows that we can reduce synchronizability wrt. an STW-bounded class to bounded model-checking.

▶ **Lemma 11.** *Let $\mathcal{S}$ be a communicating system,* com $\in \{\mathsf{p2p}, \mathsf{mb}\}$*, $k \in \mathbb{N}$, and $\mathcal{C} \subseteq \mathsf{MSC}^{k\text{-stw}}$. Then, $L_{\mathrm{com}}(\mathcal{S}) \subseteq \mathcal{C}$ iff $L_{\mathrm{com}}(\mathcal{S}) \cap \mathsf{MSC}^{(k+2)\text{-stw}} \subseteq \mathcal{C}$.*

The result follows from the following lemma. Note that a similar property was shown in [18, Proposition 5.4] for the specific class of existentially k-bounded MSCs.

▶ **Lemma 12.** *Let $k \in \mathbb{N}$ and $\mathcal{C} \subseteq \mathsf{MSC}^{k\text{-stw}}$. For all $M \in \mathsf{MSC} \setminus \mathcal{C}$, we have $(\mathit{Pref}(M) \cap \mathsf{MSC}^{(k+2)\text{-stw}}) \setminus \mathcal{C} \neq \emptyset$.*

We now have all ingredients to state a generic decidability result for synchronizability:

▶ **Theorem 13.** *Fix finite sets $\mathbb{P}$ and $\mathbb{M}$. Suppose* com $\in \{\mathsf{p2p}, \mathsf{mb}\}$ *and let $\mathcal{C} \subseteq \mathsf{MSC}$ be an MSO-definable and STW-bounded class (over $\mathbb{P}$ and $\mathbb{M}$). The following problem is decidable: Given a communicating system $\mathcal{S}$, do we have $L_{\mathrm{com}}(\mathcal{S}) \subseteq \mathcal{C}$?*

Proof. Consider the MSO-formula φ such that $L(\varphi) = \mathcal{C}$, and let $k \in \mathbb{N}$ such that $\mathcal{C} \subseteq \mathsf{MSC}^{k\text{-stw}}$. We have $L_{\mathrm{com}}(\mathcal{S}) \subseteq \mathcal{C} \overset{\text{Lemma 11}}{\iff} L_{\mathrm{com}}(\mathcal{S}) \cap \mathsf{MSC}^{(k+2)\text{-stw}} \subseteq \mathcal{C} \iff L_{\mathrm{com}}(\mathcal{S}) \cap \mathsf{MSC}^{(k+2)\text{-stw}} \subseteq L(\varphi)$. The latter can be solved thanks to Fact 9 and Theorem 10. ◀

▶ Remark 14. Note that, in some cases (cf. Section 4), $\mathbb{P}$ and $\mathbb{M}$ are part of the input and the concrete class $\mathcal{C}$ may be parameterized by a natural number so that it is part of the input, too. Then, we need to be able to compute the MSO formula characterizing the class as well as the bound on the special tree-width.

4 Application to Concrete Classes of Synchronizability

In this section, we instantiate our general framework by specific classes. Table 1 gives a summary of the results.

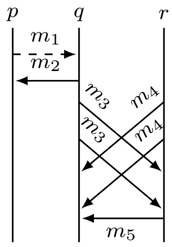


Figure 4 MSC M_2.

4.1 A New General Class: Weakly Synchronous MSCs

We first introduce the class of weakly synchronous MSCs. This is a generalization of synchronous MSCs studied earlier, in [8, 14], which we shall discuss later. We say an MSC is weakly synchronous if it is breakable into *exchanges* where an exchange is an MSC that allows one to schedule all sends before all receives. Let us define this formally:

▶ **Definition 15** (exchange). *Let $M = (\mathcal{E}, \to, \lhd, \lambda)$ be an MSC. We say that M is an* exchange *if $SendEv(M)$ is a $\leq_M$-downward-closed set.*

▶ **Definition 16** (weakly synchronous). *We say that $M \in$ MSC is* weakly synchronous *if it is of the form $M = M_1 \cdot \ldots \cdot M_n$ such that every M_i is an exchange.*

We use the term *weakly* to distinguish from variants introduced later.

▶ **Example 17.** Consider the MSC M_2 in Fig. 4. It is is weakly synchronous. Indeed, m_1, m_2, and m_5 are independent and can be put alone in an exchange. Repetitions of m_3 and m_4 are interlaced, but they constitute an exchange, as we can do all sends and then all receptions.

An easy adaptation of a characterization from [14] yields the following result for weakly synchronous MSCs:

▶ **Proposition 18.** *Let M be an MSC. Then, M is weakly synchronous iff no RS edge occurs on any cyclic path in the conflict graph* CG*(M).*

It is easily seen that the characterization from Proposition 18 is LCPDL-definable:

▶ **Corollary 19.** *The sets of weakly synchronous MSCs and weakly synchronous* mailbox *MSCs are LCPDL-definable. Both formulas have polynomial size.*

Moreover, under the mailbox semantics, we can show:

▶ **Proposition 20.** *The set of weakly synchronous mailbox MSCs is STW-bounded (in fact, it is included in* $\mathsf{MSC}^{4|\mathbb{P}|\text{-stw}}$*).*

Proof. Let M be fixed, and let us sketch Eve's winning strategy. Let $n = |\mathbb{P}|$.

The first step for Eve is to split M in exchanges. She first disconnects the first exchange from the rest of the graph ($2n$ pebbles are needed), then she disconnects the second exchange from the rest of the graph ($2n$ pebbles needed, plus n pebbles remaining from the first round), and so on for each exchange.

So we are left with designing a winning strategy for Eve with $4n + 1$ pebbles on the graph of an exchange M_0, where initially there are (at most) n pebbles placed on the first event of each process and also (at most) n pebbles placed on the last event of each process. Eve also places (at most) n pebbles on the last send event of each process and also (at most) n pebbles on the first receive event of each process. Eve erases the (at most) n $\rightarrow$-edges between the last send event and the first receive event.

We are now in a configuration that will be our invariant.

Let us fix a mailbox linearization of M_0 and let e be the first send event in this linearization.

- if e is an unmatched send of process p, Eve places her last pebble on the next send event of p (if it exists), let us call it e'. Then Eve erases the $\rightarrow$-edge (e, e'), and now e is completely disconnected, so it can be removed and the pebble can be taken back.
- if $e \lhd e'$, with e' a receive event of process q, then due to the mailbox semantics e' is the first receive event of q, so it has a pebble placed on it. Eve removes the $\lhd$-edge between e and e', then using the extra pebble she disconnects e and places a pebble on the $\rightarrow$-successor of e, then she also disconnects e' and places a pebble on the $\rightarrow$-successor of e'.

After that, we are back to our invariant, so we can repeat the same strategy with the second send event of the linearization, and so on until all edges have been erased. ◀

We obtain the following result as a corollary. Note that it assumes the mailbox semantics.

▶ **Theorem 21.** *The following problem is decidable in exponential time: Given $\mathbb{P}$, $\mathbb{M}$, and a communicating system $\mathcal{S}$ (over $\mathbb{P}$ and $\mathbb{M}$), is every MSC in $L_{\mathsf{mb}}(\mathcal{S})$ weakly synchronous?*

Proof. According to Corollary 19, we determine the LCPDL formula Φ_{wsmb} such that $L(\Phi_{\mathsf{wsmb}})$ is the set of weakly synchronous mailbox MSCs. Moreover, recall from Proposition 20 that the special tree-width of all weakly synchronous mailbox MSCs is bounded by $4|\mathbb{P}|$. By Lemma 11, $L_{\mathsf{mb}}(\mathcal{S}) \subseteq L(\Phi_{\mathsf{wsmb}})$ iff $L_{\mathsf{mb}}(\mathcal{S}) \cap \mathsf{MSC}^{(4|\mathbb{P}|+2)\text{-stw}} \subseteq L(\Phi_{\mathsf{wsmb}})$. The latter is an instance of the bounded model-checking problem. As the length of Φ_{wsmb} is polynomial in $|\mathbb{P}|$, we obtain that the original problem is decidable in exponential time by Theorem 10. ◀

For the same reasons, the model-checking problem for "weakly synchronous" systems is decidable. Interestingly, a reduction from Post's correspondence problem shows that decidability fails when adopting the p2p semantics:

▶ **Theorem 22.** *The following problem is undecidable: Given finite sets $\mathbb{P}$ and $\mathbb{M}$ as well as a communicating system $\mathcal{S}$, is every MSC in $L_{\mathsf{p2p}}(\mathcal{S})$ weakly synchronous?*

4.2 Weakly k-Synchronous MSCs

This negative result for the p2p semantics motivates the study of other classes. In fact, our framework captures several classes introduced in the literature.

▶ **Definition 23** (k-exchange)**.** *Let $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ be an MSC and $k \in \mathbb{N}$. We call M a* k-exchange *if M is an exchange and $|SendEv(M)| \leq k$.*

Let us now recall the definition from [8,14], but (equivalently) expressed directly in terms of MSCs rather than via *executions*. It differs from the weakly synchronous MSCs in that here, we insist on constraining the number of messages sent per exchange to be at most k.

▶ **Definition 24** (weakly k-synchronous)**.** *Let $k \in \mathbb{N}$. We say that $M \in \mathsf{MSC}$ is weakly k-synchronous if it is of the form $M = M_1 \cdot \ldots \cdot M_n$ such that every M_i is a k-exchange.*

▶ **Example 25.** MSC M_3 in Fig. 5 is weakly 1-synchronous, as it can be decomposed into three 1-exchanges (the decomposition is depicted by the horizontal dashed lines). We remark that $M_3 \in \mathsf{MSC}_{\mathsf{mb}}$. Note that there is a p2p linearization that respects the decomposition.

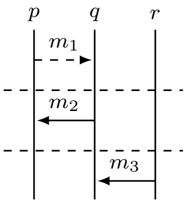


Figure 5 MSC M_3.

On the other hand, a mailbox linearization needs to reorganize actions from different MSCs: the sending of m_3 needs to be done before the sending of m_1. Note that M_1 in Fig. 1 is also weakly 1-synchronous.

▶ **Proposition 26.** *Let $k \in \mathbb{N}$. The set of weakly k-synchronous p2p (mailbox, respectively) MSCs is effectively MSO-definable.*

In fact, MSO-definability essentially follows from the following known theorem:

▶ **Theorem 27** ([14])**.** *Let M be an MSC. Then, M is weakly k-synchronous iff every SCC in its conflict graph $\mathsf{CG}(M)$ is of size at most k and no RS edge occurs on any cyclic path.*

This property is similar to the graphical characterization of weakly synchronous MSCs, except for the condition that every SCC in the conflict graph is of size at most k. Furthermore, it is easy to establish a bound on the special tree-width:

▶ **Proposition 28.** *Let $k \in \mathbb{N}$. The set of MSCs that are weakly k-synchronous have special tree-width bounded by $2k + |\mathbb{P}|$.*

Hence, we can conclude that the class of weakly k-synchronous MSCs is MSO-definable and STW-bounded. As a corollary, we get the following (known) decidability result, but via an alternative proof:

▶ **Theorem 29** ([8, 14])**.** *For $\mathsf{com} \in \{\mathsf{p2p}, \mathsf{mb}\}$, the following problem is decidable: Given finite sets $\mathbb{P}$ and $\mathbb{M}$, a communicating system $\mathcal{S}$, and $k \in \mathbb{N}$, is every MSC in $L_{\mathsf{com}}(\mathcal{S})$ weakly k-synchronous?*

Proof. We proceed similarly to the proof of Theorem 21. For the given $\mathbb{P}$, $\mathbb{M}$, and k, we first determine, using Proposition 26, the MSO formula φ_k such that $L(\varphi_k)$ is the set of weakly k-synchronous p2p/mailbox MSCs. From Proposition 28, we know that the special tree-width of all weakly k-synchronous MSCs is bounded by $2k + |\mathbb{P}|$. By Lemma 11, we have $L_{\mathsf{com}}(\mathcal{S}) \subseteq L(\varphi_k)$ iff $L_{\mathsf{com}}(\mathcal{S}) \cap \mathsf{MSC}^{(2k+|\mathbb{P}|+2)\text{-}\mathsf{stw}} \subseteq L(\varphi_k)$. The latter is an instance of the bounded model-checking problem. By Fact 9 and Theorem 10, we obtain decidability. ◀

▶ Remark 30. The set of weakly k-synchronous MSCs is not directly expressible in LCPDL (the reason is that LCPDL does not have a built-in counting mechanism). However, its *complement* is expressible in the extension of LCPDL with existentially quantified propositions (we need $k + 1$ of them). The model-checking problem for this kind of property is still in EXPTIME and, therefore, so is the problem from Theorem 29 when k is given in unary. It is very likely that our approach can also be used to infer the PSPACE upper bound from [8] by showing bounded *path width* and using finite word automata instead of tree automata. Finally, note that the problem to decide whether there exists an integer $k \in \mathbb{N}$ such that all MSCs in $L_{\mathsf{com}}(\mathcal{S})$ are weakly k-synchronous has recently been studied in [20] and requires different techniques.

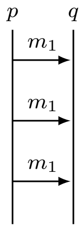


Figure 6 MSC M_4.

Observe also that we can remove the constraint of all the sends preceding all the receives in a k-exchange, and still have decidability. We then have the following definition.

▶ **Definition 31** (modified k-exchange). *Let $M = (\mathcal{E}, \rightarrow, \lhd, \lambda)$ be an MSC and $k \in \mathbb{N}$. We call M a* modified k-exchange *if $|SendEv(M)| \leq k$.*

We extend this notion to consider modified weakly k-synchronous executions as before, and the graphical characterization of this property is that there are at most k nodes in every SCC of the conflict graph. Hence, this class is also MSO-definable, and since each modified k-exchange has at most $2k$ events, it also has bounded special tree-width.

4.3 Strongly k-Synchronous MSCs and Other Classes

Our framework can be applied to a variety of other classes. Here we show how the decidability results can be shown for a variant of the class of weakly k-synchronous MSCs.

▶ **Definition 32.** *Let $M = (\mathcal{E}, \rightarrow, \lhd, \lambda) \in \mathsf{MSC_{mb}}$. We call M* strongly k-synchronous *if it can be written as $M = M_1 \cdot \ldots \cdot M_n$ such that every MSC $M_i = (\mathcal{E}_i, \rightarrow_i, \lhd_i, \lambda_i)$ is a k-exchange and, for all $(e, f) \in \sqsubset_M$, there are $1 \leq i \leq j \leq n$ such that $e \in \mathcal{E}_i$ and $f \in \mathcal{E}_j$.*

▶ **Example 33.** MSC $M_4 \in \mathsf{MSC_{mb}}$ in Fig. 6 is strongly 1-synchronous. Indeed, we can decompose it into 1-exchanges and this decomposition allows for a total order compatible with $\sqsubset_{M_4}$. Moreover, MSC M_3 in Fig. 5, which is weakly 1-synchronous, is strongly 3-synchronous. Indeed, we need to put the three messages in the same k-exchange to regain our total order. Finally, for all k, MSC M_1 in Fig. 1 is not strongly k-synchronous, as we cannot put all messages in the same k-exchange, where all sends are followed by all receptions. Here, this is not possible as the reception of m_3 has to take place before the sending of m_4.

▶ **Proposition 34.** *For all $k \in \mathbb{N}$, the set of strongly k-synchronous mailbox MSCs is MSO-definable and STW-bounded.*

The proof proceeds similarly to what has been shown in the previous cases, but MSO-definability now relies on an *extended* conflict graph. As a corollary, we thus obtain:

▶ **Theorem 35.** *The following problem is decidable: Given finite sets $\mathbb{P}$ and $\mathbb{M}$, a communicating system $\mathcal{S}$, and $k \in \mathbb{N}$, is every MSC in $L_{\mathsf{mb}}(\mathcal{S})$ strongly k-synchronous?*

▶ Remark 36. Only mailbox MSCs are considered for the definition of strongly k-synchronous MSCs for the following reason: A natural p2p analogue of Definition 32 would require from the decomposition that, for all $(e, f) \in \leq_M$, there are indices $1 \leq i \leq j \leq n$ such that $e \in \mathcal{E}_i$ and $f \in \mathcal{E}_j$. But this is always satisfied. So the natural definition of "strongly k-synchronous MSCs" would coincide with weakly k-synchronous MSCs.

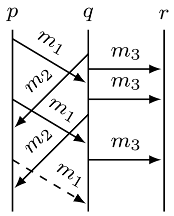


Figure 7 MSC M_5.

Like the variant for the case of weakly synchronous MSCs, we can also generalize strongly k-synchronous MSCs by removing the restriction on the number of messages per exchange:

▶ **Definition 37.** *Let $M = (\mathcal{E}, \dashrightarrow, \lhd, \lambda) \in \mathsf{MSC}_{\mathsf{mb}}$. We call M* strongly synchronous *if it can be written as $M = M_1 \cdot \ldots \cdot M_n$ such that every MSC $M_i = (\mathcal{E}_i, \dashrightarrow_i, \lhd_i, \lambda_i)$ is an exchange and, for all $(e, f) \in \sqsubset_M$, there are indices $1 \leq i \leq j \leq n$ such that $e \in \mathcal{E}_i$ and $f \in \mathcal{E}_j$.*

Similarly to the constructions for strongly k-synchronous MSCs, we can obtain a graphical characterization where we only look for the absence of RS-edges in a cycle. Hence, this class is also MSO-definable (in fact, even LCPDL-definable) and STW-bounded.

4.4 Existentially k-Bounded MSCs

Now, we turn to existentially k-bounded MSCs [18,19,24]. Synchronizability has been studied for the p2p case in [18], so we only consider the mailbox case here. A linearization $\rightsquigarrow$ of an MSC $M = (\mathcal{E}, \dashrightarrow, \lhd, \lambda) \in \mathsf{MSC}$ is called *k-mailbox-bounded* if, for all $e \in \mathit{Matched}(M)$, say with $\lambda(e) = \mathit{send}(p, q, m)$, we have $\#_{\mathit{Send}(\_,q,\_)}(\rightsquigarrow, e) - \#_{\mathit{Rec}(\_,q,\_)}(\rightsquigarrow, e) \leq k$.

▶ **Definition 38.** *Let $M = (\mathcal{E}, \dashrightarrow, \lhd, \lambda) \in \mathsf{MSC}$ and $k \in \mathbb{N}$. We call M* existentially k-mailbox-bounded *if it has some mailbox linearization that is k-mailbox-bounded.*

Note that every existentially k-mailbox-bounded MSC is a mailbox MSC.

▶ **Example 39.** MSC M_5 in Fig. 7 is existentially 1-mailbox-bounded, as witnessed by the (informally given) linearization $\mathsf{s}(q,p,m_2) \rightsquigarrow \mathsf{s}(p,q,m_1) \rightsquigarrow \mathsf{s}(q,r,m_3) \rightsquigarrow \mathsf{r}(q,r,m_3) \rightsquigarrow \mathsf{r}(p,q,m_1) \rightsquigarrow \mathsf{s}(p,q,m_1) \rightsquigarrow \mathsf{r}(q,p,m_2) \rightsquigarrow \mathsf{s}(q,r,m_3) \ldots$ Note that M_5 is neither weakly nor strongly synchronous as we cannot divide it into exchanges.

▶ **Proposition 40.** *For all $k \in \mathbb{N}$, the set of existentially k-mailbox-bounded MSCs is MSO-definable and STW-bounded.*

This extension is also valid for the p2p definition of existentially k-bounded MSCs, which were addressed in [18]. Finally, our framework can also be adapted to treat universally bounded systems [21,24].

5 Relations Between Classes

In this section we study how the classes introduced and recalled so far are related to each other. Notably, depending on the semantics (p2p or mailbox), we obtain two different classifications. The results are summed up in Figures 8 and 9. Here, we define existentially k-p2p-bounded MSCs and universally bounded counterparts as expected.

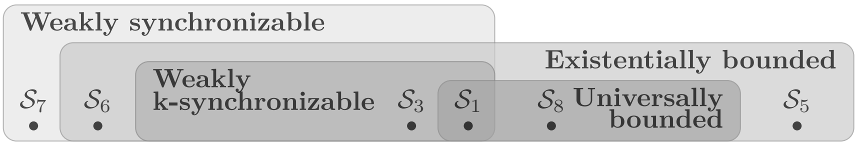


Figure 8 Hierarchy of classes for p2p systems.

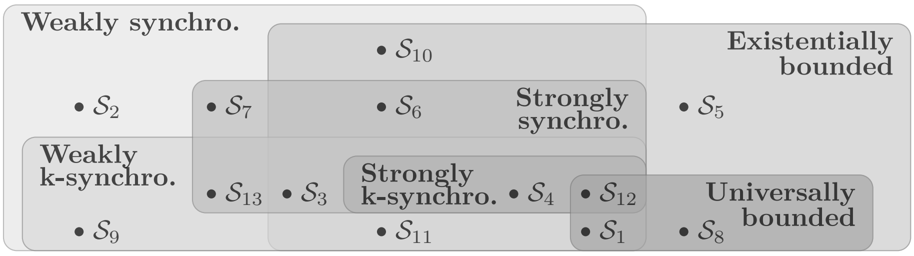


Figure 9 Hierarchy of classes for mailbox systems.

To refer to those systems we use the following terminology: a system $\mathcal{S}$ is called weakly synchronizable (resp. strongly synchronizable) if all MSCs M in the respective language are weakly synchronous (resp. strongly synchronous). A system is called weakly k-synchronizable (resp. strongly k-synchronizable, existentially bounded or universally bounded) if all MSCs are weakly k-synchronous (resp. strongly k-synchronous, existentially k-bounded or universally k-bounded). A similar comparison relating existentially bounded systems, weakly k-synchronizable systems, as well as other systems that have not been described here, can also be found in [23] for p2p systems.

We give some results showing the inclusion of certain classes. Recall that strong k-synchronizability is tailored to mailbox systems (cf. also Remark 36) so that, for p2p systems, we only consider the case of weak (k-)synchronizability.

▶ **Proposition 41.** *Every weakly k-synchronous MSC is existentially k-p2p-bounded. Moreover, every strongly k-synchronous mailbox MSC is existentially k-mailbox-bounded.*

Finally, if a system is weakly synchronizable and universally k-bounded then, there is a k' such that it is also weakly k'-synchronizable. The equivalent property is also valid for strong classes.

▶ **Proposition 42.** *Every weakly (resp. strongly) synchronizable and universally k-bounded system is weakly (resp. strongly) k'-synchronizable for a k'.*

6 Conclusion and Perspectives

We have presented a unifying framework based on MSO logic and (special) tree-width, that brings together existing definitions, explains their good properties, and allows one to easily derive other, more general definitions and decidability results for synchronizability. Let us notice that the send-synchronizability does not fit in our framework because the question $L_{\mathsf{p2p}}(\mathcal{S}) \subseteq \mathcal{C}_0$ would be decidable (by Theorem 13), where $\mathcal{C}_0$ is the set of send-synchronizable MSCs, but this property is equivalent to checking whether the system $\mathcal{S}$ is send-synchronizable and this last property is undecidable [16].

Many other related questions could be studied in the future. For example, we could think about the hypotheses to add to our general framework to make the problem *"does there exist an $k \geq 0$ such that $L_{\mathsf{p2p}}(\mathcal{S}) \subseteq \mathcal{C}_k$?"* decidable. From very recent work [20], one knows that the problem *"does there exist an $k \geq 0$ such that the system is (weakly/strongly) k-synchronizable?"* is decidable; but it remains to be seen if it would be possible to obtain these results by showing that these properties can be expressed in a decidable extension of our framework. Let us remark that the decidability of the question whether there exists an $k \geq 0$ such that $L_{\mathsf{p2p}}(\mathcal{S}) \subseteq \mathcal{C}_k$ allows us to build a bounded model checking strategy by first deciding whether there exists such an $k \geq 0$ and then by testing if $L_{\mathsf{p2p}}(\mathcal{S}) \subseteq \mathcal{C}_k$ for $k = 0, 1, 2 \ldots$. One may use this strategy for weakly/strongly synchronizable systems, but not for existentially bounded systems (except for deadlock-free systems) or for deterministic deadlock-free universally bounded systems. In [23], Lange and Yoshida introduced an *asynchronous compatibility* property and it would also be interesting to verify whether this property could be expressed into our framework.

References

1 C. Aiswarya and Paul Gastin. Reasoning about distributed systems: WYSIWYG (invited talk). In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 11–30. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.11.

2 C. Aiswarya, Paul Gastin, and K. Narayan Kumar. Verifying communicating multi-pushdown systems via split-width. In *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014*, volume 8837 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2014.

3 Samik Basu and Tevfik Bultan. Choreography conformance via synchronizability. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th International Conference on World Wide Web, WWW 2011, Hyderabad, India, March 28 - April 1, 2011*, pages 795–804. ACM, 2011. doi:10.1145/1963405.1963516.

4 Bernard Boigelot and Patrice Godefroid. Symbolic verification of communication protocols with infinite state spaces using qdds (extended abstract). In Rajeev Alur and Thomas A. Henzinger, editors, *Computer Aided Verification, 8th International Conference, CAV '96, New Brunswick, NJ, USA, July 31 - August 3, 1996, Proceedings*, volume 1102 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1996. doi:10.1007/3-540-61474-5_53.

5 Benedikt Bollig, Alain Finkel, and Amrita Suresh. Bounded reachability problems are decidable in FIFO machines. In Igor Konnov and Laura Kovacs, editors, *Proceedings of the 31st International Conference on Concurrency Theory (CONCUR'20)*, volume 171 of *Leibniz International Proceedings in Informatics*, pages 49:1–49:17, Vienna, Austria, 2020. Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CONCUR.2020.49.

6 Benedikt Bollig, Marie Fortin, and Paul Gastin. Communicating finite-state machines, first-order logic, and star-free propositional dynamic logic. *J. Comput. Syst. Sci.*, 115:22–53, 2021.

7 Benedikt Bollig and Paul Gastin. Non-sequential theory of distributed systems. *CoRR*, abs/1904.06942, 2019. arXiv:1904.06942.

8 Ahmed Bouajjani, Constantin Enea, Kailiang Ji, and Shaz Qadeer. On the completeness of verifying message passing programs under bounded asynchrony. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, volume 10982 of *Lecture Notes in Computer Science*, pages 372–391. Springer, 2018. doi:10.1007/978-3-319-96142-2_23.

9 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983. doi:10.1145/322374.322380.

10 Gérard Cécé and Alain Finkel. Verification of programs with half-duplex communication. *Information and Computation*, 202(2):166–190, 2005. doi:10.1016/j.ic.2005.05.006.

11 Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable channels are easier to verify than perfect channels. *Information and Computation*, 124(1):20–31, 1996. URL: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PS/CFP-IC96.ps.

12 Bruno Courcelle. Special tree-width and the verification of monadic second-order graph properties. In *FSTTCS*, volume 8 of *LIPIcs*, pages 13–29, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.13.

13 Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 547–561. Springer, 2012. doi:10.1007/978-3-642-32940-1_38.

14 Cinzia Di Giusto, Laetitia Laversa, and Étienne Lozes. On the k-synchronizability of systems. In Jean Goubault-Larrecq and Barbara König, editors, *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Proceedings*, volume 12077 of *Lecture Notes in Computer Science*, pages 157–176. Springer, 2020. doi:10.1007/978-3-030-45231-5_9.

15 Javier Esparza, Pierre Ganty, and Rupak Majumdar. A perfect model for bounded verification. In *Proceedings of the 2012 27th Annual IEEE/ACM Symposium on Logic in Computer Science*, LICS '12, pages 285–294, Washington, DC, USA, 2012. IEEE Computer Society. doi:10.1109/LICS.2012.39.

16 Alain Finkel and Étienne Lozes. Synchronizability of communicating finite state machines is not decidable. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 122:1–122:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

17 Alain Finkel and M. Praveen. Verification of Flat FIFO Systems. *Logical Methods in Computer Science*, 20(4), 2020. doi:10.23638/LMCS-16(4:4)2020.

18 Blaise Genest, Dietrich Kuske, and Anca Muscholl. On communicating automata with bounded channels. *Fundamenta Informaticae*, 80(1-3):147–167, 2007.

19 Blaise Genest, Anca Muscholl, and Dietrich Kuske. A kleene theorem for a class of communicating automata with effective algorithms. In Cristian Calude, Elena Calude, and Michael J. Dinneen, editors, *Developments in Language Theory, 8th International Conference, DLT 2004, Auckland, New Zealand, December 13-17, 2004, Proceedings*, volume 3340 of *Lecture Notes in Computer Science*, pages 30–48. Springer, 2004. doi:10.1007/978-3-540-30550-7_4.

20 Cinzia Di Giusto, Laetitia Laversa, and Étienne Lozes. Guessing the buffer bound for k-synchronizability. In *Implementation and Application of Automata - 25th International Conference, CIAA 2021, Proceedings*, Lecture Notes in Computer Science. Springer, 2021. To appear.

21 Jesper G. Henriksen, Madhavan Mukund, K. Narayan Kumar, Milind Sohoni, and P.S. Thiagarajan. A theory of regular msc languages. *Information and Computation*, 202(1):1–38, 2005.

22 Dietrich Kuske and Anca Muscholl. Communicating automata, 2014.

23 Julien Lange and Nobuko Yoshida. Verifying asynchronous interactions via communicating session automata. *CoRR*, abs/1901.09606, 2019. arXiv:1901.09606.

24 Markus Lohrey and Anca Muscholl. Bounded MSC communication. In Mogens Nielsen and Uffe Engberg, editors, *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002. Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2002 Grenoble, France, April 8-12, 2002, Proceedings*, volume 2303 of *Lecture Notes in Computer Science*, pages 295–309. Springer, 2002. doi:10.1007/3-540-45931-6_21.

25 Markus Lohrey and Anca Muscholl. Bounded MSC communication. *Inf. Comput.*, 189(2):160–181, 2004. doi:10.1016/j.ic.2003.10.002.

26 P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. In Thomas Ball and Mooly Sagiv, editors, *Proceedings of the 38th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, TX, USA, January 26-28, 2011*, pages 283–294. ACM, 2011.

27 Robert S. Streett. Propositional dynamic logic of looping and converse. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*, pages 375–383. ACM, 1981.

28 Gregor von Bochmann. Communication protocols and error recovery procedures. *Operating Systems Review*, 9(3):45–50, 1975.

Guard Automata for the Verification of Safety and Liveness of Distributed Algorithms

Nathalie Bertrand
Université Rennes, Inria, CNRS, IRISA, France

Bastien Thomas
Université Rennes, Inria, CNRS, IRISA, France

Josef Widder
Informal Systems, Wien, Austria

Abstract

Distributed algorithms typically run over arbitrary many processes and may involve unboundedly many rounds, making the automated verification of their correctness challenging. Building on domain theory, we introduce a framework that abstracts infinite-state distributed systems that represent distributed algorithms into finite-state *guard automata*. The soundness of the approach corresponds to the Scott-continuity of the abstraction, which relies on the assumption that the distributed algorithms are *layered*. Guard automata thus enable the verification of safety and liveness properties of distributed algorithms.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Theory of computation → Distributed algorithms

Keywords and phrases Verification, Distributed algorithms, Domain theory

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.15

Related Version *Full Version*: `https://hal.inria.fr/hal-03283388`

Funding This project has received funding from Interchain Foundation (Switzerland).

1 Introduction

Under the umbrella of *parameterized verification*, the verification of systems formed of an arbitrary number of agents executing the same code, has attracted quite some attention in the recent years, see for instance [18, 9]. Application examples range from distributed algorithms (e.g., for clock synchronization [28] or robot coordination [27]), cache-coherence protocols [25, 1], to chemical or biological systems [10]. In all cases, the systems are designed to operate correctly independently of the number of agents.

More specifically, *distributed algorithms* are central to various emblematic applications, including telecommunications, scientific computing, and Blockchain. Automatically proving the correctness of distributed algorithms is a particularly relevant, as stated by Lamport: "Model-checking algorithms prior to submitting them for publication should become the norm" [22]. The task, that the verification community has started to address, is quite challenging, since it aims at validating at once all instances of the algorithm for arbitrarily many processes.

Distributed algorithms with *threshold guards* are omni-present in solutions for consensus and agreement problems. Typically, these guards also are parameterized, e.g., if the number of processes in a distributed system is n, then it is natural to require that certain actions are taken only if a majority of processes is ready to do so; this results in a parameterized threshold expression of $n/2$. Due to Blockchain and other current applications these kinds of distributed algorithm enjoy recent attention from the algorithm design community as well as

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 15; pp. 15:1–15:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the verification community. the algorithm design community has been studying them for a long time, (see e.g., [11]) and typically provides hand-written proofs based on mathematical models without formal semantics.

For computer-aided verification the first challenge is to develop appropriate modeling formalisms that maintain all behaviors of the original algorithms on the one hand, and on the other hand are abstract and succinct to allow for efficient verification. Several approaches towards efficient verification have recently been proposed.

The threshold automata framework [20] targets asynchronous distributed algorithms with threshold guards and reductions (similar to [23, 17]) have been used to show that SMT-based bounded model checking is complete [19]. Later this framework was generalized and generalizations were analyzed regarding decidability [21], and complexity [5]. The current paper also targets threshold distributed algorithms, yet eventually provides an even coarser abstraction to represent their behaviors, thus reducing the overall verification complexity. Moreover, the semantics of distributed algorithms and the soundness of the abstraction rely on domain theory concepts, thus providing a solid mathematical framework to our work. Last but not least, our approach can handle infinite behaviours, in contrast to the threshold automata framework.

The logical fragment of the IVy toolset has also been shown to allow to model threshold guards by axiomising their semantics as quorum systems [7]. For instance, the reason for waiting for quorums of more than $n/2$ messages is that any two such quorums must intersect at one sender. IVy allows to express these quorum axioms and reduce verification to decidable fragments. Similar intuitions underlie verification results in the heard-of model (HO model) [13]. This computational model for distributed algorithms already targets a high level of abstractions that are sound for communication closed distributed algorithms [12]. Here a consensus logic was introduced in [16] that could be used for deductive verification and cut-off results where provided in [24] that reduce the parameterized verification problem to small finite instances. Compared to this line of work, the distributed algorithms we target share some similarities with these round-based communication closed models. Recently, a threshold automata framework for round-based algorithms was introduced that also uses a small counterexample property for verification in [29]. In contrast, we use domain theory, and particularly Scott continuity to be able to reason on infinite behaviors and thus to capture algorithms that do not necessarily terminate.

Other less related verification frameworks also target distributed algorithms with quite different techniques such as event B [26], array systems [4] or logic and automata theory [3].

Contributions

Using basic domain theory concepts, we provide a rigorous framework to model and verify (asynchronous) distributed algorithms. Our methodology applies to distributed algorithms that are structured in *layers* (that can be seen as a fine-grain notion of rounds), and may consist of countably many layers, thus capturing round-based distributed algorithms (with no *a priori* bound on the number of rounds).

- In Section 2, we define partially ordered transition systems, which serve to express the semantics our models.
- Section 3 introduces the low-level model of layered distributed systems to represent threshold based distributed algorithms. The state-space of layered distributed systems being infinite (and even not necessarily finitely representable), we provide several abstraction steps, up to a so-called guard abstraction. The soundness of each step is justified by

the Scott-continuity of the corresponding abstraction. Some steps are also complete, and thus do not introduce spurious behaviors.

- Finally, towards practical verification, we define in Section 4 the guard automaton, a finite-state abstraction of (cyclic) layered distributed systems. It overapproximates the set of infinite behaviors of distributed algorithms, and thus enabling the verification of safety as well as liveness properties. Its construction can be automated with the help of an SMT solver, paving the way to the automated verification of round-based threshold distributed algorithms.

2 A Fistful of Domain Theory

2.1 Mathematical Preliminaries

This section presents mathematical notions as well as notations that are used throughout the paper. In particular, it introduces partially ordered sets and Scott topology. The interested reader is referred to [2] for an thorough introduction to domain theory.

Sets and multisets. A *multiset* over a set X is an element of $\mathbb{N}^X$. Addition and inclusion over multisets are defined in a natural way. For $\xi, \xi' \in \mathbb{N}^X$ two multisets, $\xi + \xi' \in \mathbb{N}^X$ is the multiset such that for every $x \in X$, $(\xi + \xi')(x) = \xi(x) + \xi'(x)$. We write $\xi \sqsubseteq \xi'$ if for every $x \in X$, $\xi(x) \leq \xi'(x)$. Standard sets can be seen as special cases of multisets with the canonical bijection between the set of subsets of X (2^X) and the set of functions from X to $\{0, 1\}$.

Sequences. For X a set and $n \in \mathbb{N}$ a natural number, a sequence of elements of X of length n is some $u \in X^{\{0,\dots,n-1\}}$. Its length is $|u| = n$ and for $i < n$, $u(i) \in X$ denotes the letter at index i. $X^* = \bigcup_{n \in \mathbb{N}} X^{\{0,\dots,n-1\}}$ (resp. $X^+ = \bigcup_{n>0} X^{\{0,\dots,n-1\}}$) denotes the set of all *finite* (resp. finite and non-empty) sequences of elements of X. Moreover, $\overline{X^*} = X^* \cup X^{\mathbb{N}}$ is the set of finite or *infinite* sequences of X. For $u \in X^*$ a finite sequence and $v \in \overline{X^*}$ a finite or infinite sequence, we write $u \cdot v$ for the *concatenation* of u and v. For u and w two sequences, we write $u \prec w$ and say that u is a *prefix* of w if either w is finite and there exists $v \in \overline{X^*}$ such that $u \cdot v = w$ or $u = w$. For w a sequence and $i \leq |w|$, w_i is the prefix w of length i.

Closures and bounds for partially ordered sets. Let $(X, \sqsubseteq)$ be a partially ordered set, and $\xi \subset X$. The *upward-closure* of ξ is $\uparrow\xi = \{x \in X \mid \exists x' \in \xi, x' \sqsubseteq x\}$, and ξ is *upward-closed* if $\uparrow\xi = \xi$. Dually, one defines the *downward-closure* $\downarrow\xi$ and *downward-closed* sets. An element $x \in X$ is an *upper-bound* of ξ if for any element $x' \in \xi$, $x' \sqsubseteq x$. We write $\mathsf{ub}(\xi)$ for the set of upper-bounds of ξ. If it exists (it is then unique), the *greatest* element of ξ is $x \in X$ such that $x \in \xi$ and $x \in \mathsf{ub}(\xi)$. Dually, one defines the notion of *least* element by reversing the order. If it exists, the *least upper bound* of ξ is the least element of $\mathsf{ub}(\xi)$, and we denote it by $\bigsqcup \xi$. Finally ξ is *directed* if it is non-empty and if for every two elements $x, x' \in \xi$, $\mathsf{ub}(\{x, x'\}) \cap \xi \neq \emptyset$; intuitively, any finite subset of ξ has an upper-bound in ξ. An interesting particular case of directed case are completely ordered sets which are called *chains* in this context.

Directed Complete Partially ordered sets (DCPO). A DCPO is a partially ordered set $(X, \sqsubseteq)$ such that any directed subset $\xi \subset X$ has a (unique) least upper bound. These partially ordered sets are particularly important in semantics of programming languages.

The Scott Topology on DCPO. Directed complete partial orders are naturally equipped with the Scott topology. A subset ξ of a DCPO $(X, \sqsubseteq)$ is *Scott-closed* if it is *downward-closed* and if for any directed subset $\xi' \subset \xi$, $\bigsqcup \xi' \in \xi$. A subset is *Scott-open* if its complement in X is Scott-closed. Functions that are continuous for the Scott topology are called *Scott-continuous*. A function $f : X \to Y$ is *monotonous* if for any $x, x' \in X$, if $x \sqsubseteq x'$ then $f(x) \sqsubseteq f(x')$. A Scott-continuous function is always monotonous. A function $f : X \to Y$ is Scott-continuous if and only if for any directed subset $\xi \subset X$, $f(\bigsqcup(\xi)) = \bigsqcup(f(\xi))$. In this paper, a *partial* function $f : X \to Y$ is called *Scott-continuous* if its domain $\mathsf{dom}(f)$ is Scott-closed and if for any directed subset $\xi \subset \mathsf{dom}(f)$, $f(\bigsqcup \xi) = \bigsqcup f(\xi)$.

2.2 Partially Ordered Transition Systems

Building on domain theory, this section introduces a generic model for distributed transition systems, that will capture the semantics of distributed algorithms. An ordering naturally appears on sets of sent messages –that can only grow– and the asynchrony requires the order to be partial only.

▶ **Definition 1.** *A* partially ordered transition system *(POTS) is a tuple $\mathcal{O} = (X, \sqsubseteq, A)$ where:*
- *$(X, \sqsubseteq)$ forms a DCPO.*
- *A is a set of partial functions, called* actions*, from X to itself and such that for every $a \in A$ and every $x \in \mathsf{dom}(a)$, $x \sqsubseteq a(x)$.*

▶ **Definition 2.** *A* schedule *is a (finite or infinite) sequence of* actions*: $\sigma = (a_t)_{t<T}$, with $T \in \overline{\mathbb{N}}$. A schedule $\sigma = (a_t)_{t<T}$ is* applicable *at $x \in X$ if there exists a sequence $(x_t)_{t<T+1}$ with $x_0 = x$, and for every $t < T$, $x_t \in \mathsf{dom}(a_t)$ and $a_t(x_t) = x_{t+1}$. In this case, we write* $\mathsf{configs}(x, \sigma)$ *for the sequence $(x_t)_{t<T+1}$, and $x \star \sigma$ for $\bigsqcup\{x_t \mid t < T{+}1\}$.*

The above definition uses the convention that $\infty{+}1 = \infty$. Note that if σ is applicable at x, then the sequence $(x_t)_{t<T+1}$ is unique. Moreover, the least upper bound $\bigsqcup\{x_t \mid t < T + 1\}$ exists because for any $t < T$, $x_t \sqsubseteq x_{t+1}$ and $\{x_t \mid t < T + 1\}$ is therefore a chain. When $\sigma = (a_t)_{t<T}$ is finite, $x \star \sigma = x \star a_0 \star \cdots \star a_{T-1}$ denotes the last element of the monotonous sequence $\mathsf{configs}(x, \sigma)$. In particular, for $a \in A$ and $x \in \mathsf{dom}(a)$, $x \star a = a(x)$. When $\sigma_t \in A^t$ is defined as the prefix of length t of σ, $x_t = x \star \sigma_t$ and it follows: $x \star \sigma = \bigsqcup\{x \star \sigma_t \mid t < T, t \in \mathbb{N}\}$.

The following lemma will be useful throughout the paper:

▶ **Lemma 3.** *For $x \in X$, the set $\mathrm{App}(x)$ of schedules applicable at x is Scott-closed for the prefix ordering and the function: $[x \star \_] : \mathrm{App}(x) \to X$ is Scott-continuous.*

▶ **Definition 4.** *An* abstraction *between POTS $\mathcal{O} = (X, \sqsubseteq, A)$ and $\mathcal{O}' = (X', \sqsubseteq, A')$ consists of*
- *a* set abstraction $\mathsf{ab}_X : X \to X'$ *which is a Scott-continuous function;*
- *a* monoid abstraction $\mathsf{ab}_A : A^* \to A'^*$ *which is a monoid morphism (with slight abuse of notation, ab_A also denotes its Scott-continuous extension $\overline{A^*} \to \overline{A'^*}$);*

both such that for every $a \in A$ and every $x \in \mathsf{dom}(a)$, $\mathsf{ab}_A(a) \in A'^*$ is applicable at $\mathsf{ab}_X(x) \in X'$ and $\mathsf{ab}_X(x \star a) = \mathsf{ab}_X(x) \star \mathsf{ab}_A(a)$.

The last condition of the definition of abstraction translates into the commutativity of the diagram in Figure 1a. The soundness of the abstraction for any (possibly infinite) schedule is stated in the following proposition and illustrated on Figure 1b.

▶ **Proposition 5.** *Let $(\mathsf{ab}_X, \mathsf{ab}_A)$ be an abstraction between $\mathcal{O} = (X, \sqsubseteq, A)$ and $\mathcal{O}' = (X', \sqsubseteq, A')$, $x \in X$ be an element, and $\sigma \in \overline{A^*}$ a schedule. If σ is applicable at x, then $\mathsf{ab}_A(\sigma)$ is applicable at $\mathsf{ab}_X(x)$ and $\mathsf{ab}_X(x \star \sigma) = \mathsf{ab}_X(x) \star \mathsf{ab}_A(\sigma)$.*

$$\begin{tikzcd}
\mathsf{dom}(a) \subset X \arrow[r, "a"] \arrow[d, "\mathsf{ab}_X"'] & X \arrow[d, "\mathsf{ab}_X"] \\
\mathsf{dom}(\mathsf{ab}_A(a)) \subset X' \arrow[r, "\mathsf{ab}_A(a)"'] & X'
\end{tikzcd}$$

(a) By Definition 4 diagram commutes for any action $a \in A$.

$$\begin{tikzcd}
X \arrow[r, "a_0"] \arrow[d, "\mathsf{ab}_X"'] \arrow[rrr, bend left, "\sigma"] & X \arrow[r, "a_1"] \arrow[d, "\mathsf{ab}_X"'] & \cdots & X \arrow[d, "\mathsf{ab}_X"'] \\
X' \arrow[r, "\mathsf{ab}_A(a_0)"] \arrow[rrr, bend right, "\mathsf{ab}_A(\sigma)"'] & X' \arrow[r, "\mathsf{ab}_A(a_1)"] & \cdots & X'
\end{tikzcd}$$

(b) By Proposition 5 diagram commutes for any schedule σ.

Figure 1 $(\mathsf{ab}_X, \mathsf{ab}_A)$ forms an abstraction between the POTS $(X, \sqsubseteq, A)$ and $(X', \sqsubseteq, A')$.

The proof of this proposition is by transfinite induction on the length of schedules: showing that the result holds for finite schedules is easy, and continuity arguments (such as Lemma 3) are then used to extend to infinite schedules.

3 Layered Distributed Systems and their Abstractions

This section introduces a low-level model for distributed algorithms, whose semantics will be expressed as a POTS. The model is structured in layers, thus restricting the application to algorithms with a specific shape. However, many distributed algorithms from the literature fall in this class, and minor modifications of other algorithms make them amenable to our techniques. The restriction to layered models is used several times in the theoretical developments that follow.

3.1 Layered Distributed Transition Systems

This section introduces *Layered Distributed Transition Systems* (LDTSs) as a model for distributed algorithms, such as the Phase King algorithm [8]. A simplified version of the algorithm is provided in Algorithm 1. This algorithm operates in rounds, each consisting of three steps:

- Broadcast a message (ℓ, m) to all process where ℓ is the round index (line 3)
- Receive the messages $(\ell, \_)$ sent in this round (line 4)
- Update the process variables according to the received messages (lines 5 to 12)

In general, such a series of three instructions, indexed by $\ell \in \mathbb{N}$, is called a *layer* and it refines the classical notion of *rounds*: for instance, in Ben-Or's consensus algorithm [6], each round comprises two layers. Note that layers are assumed to be *communication-closed* [17, 14]: the update instruction at layer ℓ only depends on received messages from the same layer.

Distributed algorithms run over a finite set of *processes*, and at every point in time, the local state of a process is defined by the valuation of its local variables. In this paper, the contents of a sent message is not particularly relevant as it can be deduced from the local state of its sender. Therefore, the communications can be encoded by guards that prevent a process from taking a transition if a condition on the state of *other processes* is not met. Formally, the syntax of layered distributed transition systems is as follows:

Algorithm 1 Inspired by the Phase King Algorithm, this algorithm is a *synchronous* algorithm targetting the resolution of binary consensus. It executes $t{+}1$ rounds. In round $\ell \in \{0 \dots t\}$, the local value v of each process is updated either according to the majority, or to the value of the process with id ℓ (the King process).

```
Process PhaseKing($n, t, \mathrm{id}, v$):
    Data: $n$ processes, $t < \frac{n}{4}$ Byzantine faults, $\mathrm{id} \in \{0 \dots n-1\}$, $v \in \{0, 1\}$.
    for $\ell = 0$ to $t$ do
        broadcast $(\ell, \mathrm{id}, v)$
        receive all the messages $(\ell, \_, \_)$
        $n_0 \leftarrow$ number of messages $(\ell, \_, 0)$ received
        $n_1 \leftarrow$ number of messages $(\ell, \_, 1)$ received
        if $n_0 > \frac{n}{2} + t$ then
            $v \leftarrow 0$
        else if $n_1 > \frac{n}{2} + t$ then
            $v \leftarrow 1$
        else
            $v \leftarrow v'$ where $(\ell, \ell, v')$ is a received message
    end
    return $v$;
```

▶ **Definition 6.** *A* layered distributed transition system *(LDTS) is a tuple* $\mathcal{D} = (P, S, \mathsf{guard})$ *where:*

- P *is a* finite *set of* processes
- S *is a set of* states *partitioned in* layers*:* $S = \bigcup_{\ell \in \mathbb{N}} S_\ell$*.*
  For $\perp$ *a new element, set* $S^\perp = S \cup \{\perp\}$ *and for* $\ell \in \mathbb{N}$*,* $S_\ell^\perp = S_\ell \cup \perp$*.*
  The set $S^\perp$ *is partially ordered with* $s \sqsubseteq s'$ *if* $s = \perp$ *or* $s = s'$*.*
- $\mathsf{guard} : S^2 \to 2^{[P \to S^\perp]}$ *associates to each pair of states a guard.*
  Additionally, the following layered hypothesis *is imposed:*
  For $\ell \in \mathbb{N}$*,* $s \in S_\ell$ *and* $s' \in S$*,* $\mathsf{guard}(s, s') \in 2^{[P \to S_\ell^\perp]}$*, and if* $s' \notin S_{\ell+1}$*, then* $\mathsf{guard}(s, s') = \emptyset$*.*

Intuitively, for $\ell \in \mathbb{N}$, S_ℓ is the set of states a process can be in at layer ℓ, and $\perp$ is used to represent that a process has not reached that layer yet. Although trivial, the ordering on $S^\perp$ shows sufficient to represent the semantics of distributed algorithms. Moreover, the *guards* correspond to a condition on messages *received* from other processes. Having $x \in \mathsf{guard}(s, s')$ with $x(p) = \perp$ means that there are no conditions on the messages received from process p, so that a process in state s can go to s' even if it has not received any message from p.

To define the semantics of LDTS, recall that the system *a priori* runs fully asynchronously, so that processes may be in different layers[1]. However, messages may be received by processes even if the sender has later reached a layer. This means that the state of each process at each layer should be recorded in the semantics of a LDTS. An agglomeration of local states is called a *configuration.* A *full configuration* additionally stores the messages *received* by each process, as formalized below:

▶ **Definition 7.** *Let* $\mathcal{D} = (P, S, \mathsf{guard})$ *be an LDTS. A* full configuration *of* $\mathcal{D}$ *is a pair* $c^f = (\mathsf{state}(c^f), \mathsf{received}(c^f))$ *where*

- $\mathsf{state}(c^f) : P \to \overline{S^+}$ *is such that for every* $p \in P$ *and* $\ell \in \mathbb{N}$
  - *if* $\ell < \left|\mathsf{state}(c^f)(p)\right|$*, then* $\mathsf{state}(c^f)(p)(\ell) \in S_\ell$ *and the latter is the state of* p *in* ℓ*;*
  - *if* $\ell \geq \left|\mathsf{state}(c^f)(p)\right|$*, then* $\mathsf{state}(c^f)(p)(\ell) = \perp \in S_\ell^\perp$*.*
- $\mathsf{received}(c^f) : P \to P \to \mathbb{N} \to S^\perp$ *such that for every* $p \in P$*,* $\mathsf{received}(c^f)(p) \sqsubseteq \mathsf{state}(c^f)$*.*

[1] Synchronous systems can also be represented by LDTS, as illustrated with the Phase King algorithm.

The set of full configurations is denoted C^f. It is partially ordered with $\sqsubseteq$ defined by $c^f \sqsubseteq {c^f}'$ if $\mathsf{state}(c^f) \sqsubseteq \mathsf{state}({c^f}')$ *pointwise with the prefix ordering on $\overline{S^+}$ and* $\mathsf{received}(c^f) \sqsubseteq \mathsf{received}({c^f}')$ *pointwise.*

Note that $S^\perp$ is a DCPO since each of its directed subsets is finite. C^f is isomorphic to the Cartesian product $\left[(P,=) \to (\overline{S^+}, \prec)\right] \times \left[(P^2 \times \mathbb{N}, =) \to (S^\perp, \sqsubseteq)\right]$ and is therefore a DCPO too.

At a full configuration $c^f \in C^f$, two types of *actions* may happen, corresponding to receptions and internal transitions. First, a process $p \in P$ may receive a message that was sent in layer $\ell \in \mathbb{N}$ by a process $p' \in P$; this action is denoted $\mathsf{rec}\,(p, \ell, p')$. Second, a process $p \in P$ may move from a state $s \in S_\ell$ to state $s' \in S_{\ell+1}$, denoted $\mathsf{tr}\,(p, s, s')$. The effect of actions on full configurations is formally defined as follows:

▶ **Definition 8.** *The set of* actions *of an LDTS $\mathcal{D} = (P, S, \mathsf{guard})$ is*

$$A^f = \{\mathsf{rec}\,(p,p',\ell) \mid p,p' \in P, \ell \in \mathbb{N}\} \cup \bigcup_{\ell \in \mathbb{N}} \{\mathsf{tr}\,(p,s,s') \mid p \in P, s \in S_\ell, s' \in S_{\ell+1}\} \quad .$$

For $c^f \in C^f$ and $\mathsf{rec}\,(p,p',\ell) \in A^f$, the full configuration ${c^f}' = \mathsf{rec}\,(p,p',\ell)\,(c^f)$ is defined by:

- $\mathsf{state}({c^f}') = \mathsf{state}(c^f)$
- $\mathsf{received}({c^f}')(p)(p')(\ell) = \mathsf{state}(c^f)(p')(\ell)$ *and* $\mathsf{received}({c^f}')$ *equals* $\mathsf{received}(c^f)$ *elsewhere.*

For $c^f \in C^f$ and $\mathsf{tr}\,(p,s,s') \in A^f$, writing $\ell = \left|\mathsf{state}(c^f)(p)\right| - 1$, then $\mathsf{tr}\,(p,s,s')$ is enabled *at $c^f \in C^f$ if: $\ell < \infty$, $\mathsf{state}(c^f)(p)(\ell) = s$ and $\mathsf{received}(c^f)(p)(\_)(\ell) \in \mathsf{guard}(s)(s')$. In this case, the full configuration ${c^f}' = \mathsf{tr}\,(p,s,s')\,(c^f)$ is defined with:*

- $\mathsf{state}({c^f}')(p) = \mathsf{state}(c^f)(p) \cdot s'$ *and* $\mathsf{state}({c^f}')$ *equals* $\mathsf{state}(c^f)$ *elsewhere.*
- $\mathsf{received}({c^f}') = \mathsf{received}(c^f)$

Note that the reception actions are always enabled. So defined, the semantics of an LDTS is a POTS $\mathcal{O}^f_{\mathcal{D}} = \left(C^f, \sqsubseteq, A^f\right)$; in particular, the notions of schedules and abstractions apply.

▶ **Example 9.** Consider the Phase King algorithm run by three correct processes and a Byzantine one. The Byzantine process is not represented explicitly ($P = \{p_0, p_1, p_2\}$ only contains correct processes) but the guards of the LDTS account for the messages it may send. Also, the King is chosen at each round non-deterministically, abstracting process ids.

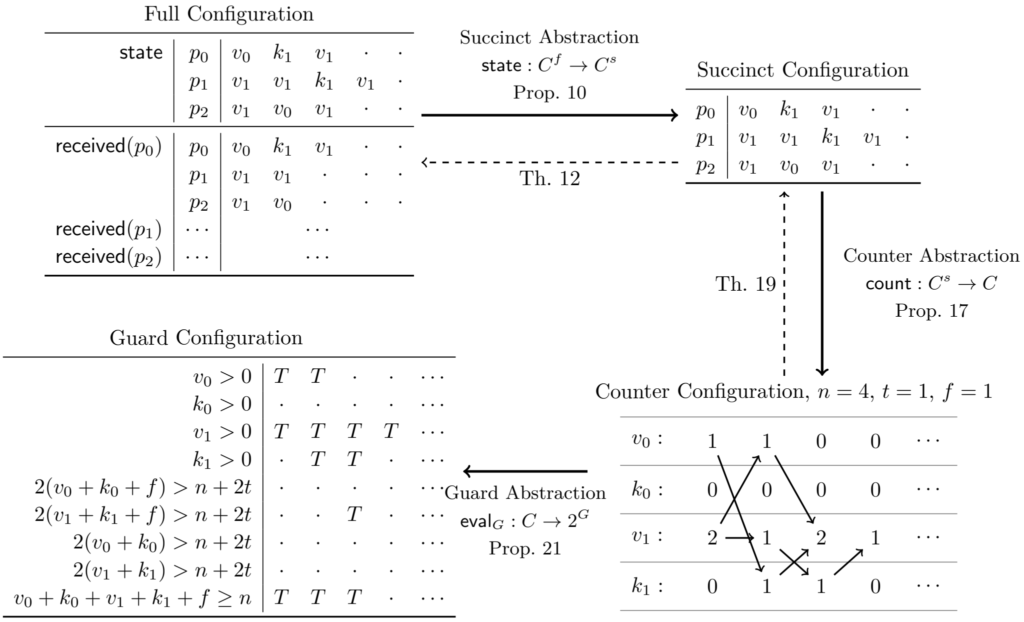
A correct process in layer ℓ may be in one of four states $S_\ell = \{v_0, v_1, k_0, k_1\}$, where k_x (resp. v_x) represents that the local value of v is $x \in \{0,1\}$ and that the process is currently King (resp. not King). A full configuration, say c^f, is depicted top-left of Figure 2. The sequence states process p_0 went through so far is $\mathsf{state}(c^f)(p_0) = v_0 \cdot k_1 \cdot v_1$. Also, $\mathsf{received}(c^f)(p_0)(p_2)(0) = v_1$ represents that process p_0 received the message that process p_2 was in state v_1 at layer 0. In contrast, p_0 does not know the state of p_2 at layer 2 (represented by a blank space instead of $\perp$ for commodity). Thus, in c^f, the message sent by process p_2 at layer 2 has yet to be received by p_0. The action $\mathsf{rec}\,(p_0, p_2, 2)$ corresponding to this reception is therefore enabled at c^f. The resulting configuration $c^f \star \mathsf{rec}\,(p_0, p_2, 2)$ would be identical to c^f except for $\mathsf{received}(c^f \star \mathsf{rec}\,(p_0,p_2,2))(p_0)(p_2)(2) = \mathsf{state}(c^f)(p_2)(2) = v_1$ instead of $\perp$. The reception $\mathsf{rec}\,(p_0, p_1, 2)$ can also happen at $c^f \star \mathsf{rec}\,(p_0, p_2, 2)$. The resulting configuration ${c^f}' = c^f \star \mathsf{rec}\,(p_0,p_2,2) \star \mathsf{rec}\,(p_0,p_1,2)$ coincides with c^f except for

$$\mathsf{received}({c^f}')(p_0) = \begin{array}{llll} p_0: & v_0 & k_1 & v_1 \\ p_1: & v_1 & v_1 & k_1 \\ p_2: & v_1 & v_0 & v_1 \end{array}$$

Now p_0 has received more than $\frac{n}{2} + t$ messages in $\{v_1, k_1\}$ so that it updates its value to 1 in the next round. Therefore, the action $\mathsf{tr}\,(p_0, v_1, v_1)$ is enabled at ${c^f}'$ and the configuration ${c^f}' \star \mathsf{tr}\,(p_0, v_1, v_1)$ is equal to ${c^f}'$ except for $\mathsf{state}\big({c^f}' \star \mathsf{tr}\,(p_0, v_1, v_1)\big) = v_0 \cdot k_1 \cdot v_1 \cdot v_1$.

3.2 Abstracting Received Messages

The partially ordered transition system $\mathcal{O}^f_{\mathcal{D}}$ is fine-grained and rather complex to analyze, therefore the aim of the rest of this section is to define simpler POTS, that preserve or overapproximate the semantics of $\mathcal{O}^f_{\mathcal{D}}$. The successive steps are represented in Figure 2.



Full Configuration

state	p_0	v_0	k_1	v_1	·	·
	p_1	v_1	v_1	k_1	v_1	·
	p_2	v_1	v_0	v_1	·	·
received(p_0)	p_0	v_0	k_1	v_1	·	·
	p_1	v_1	v_1	·	·	·
	p_2	v_1	v_0	·	·	·
received(p_1)	···			···		
received(p_2)	···			···		

Succinct Configuration

p_0	v_0	k_1	v_1	·	·
p_1	v_1	v_1	k_1	v_1	·
p_2	v_1	v_0	v_1	·	·

Counter Configuration, $n = 4$, $t = 1$, $f = 1$

v_0 :	1	1	0	0	···
k_0 :	0	0	0	0	···
v_1 :	2	1	2	1	···
k_1 :	0	1	1	0	···

Guard Configuration

$v_0 > 0$	T	T	·	·	···
$k_0 > 0$	·	·	·	·	···
$v_1 > 0$	T	T	T	T	···
$k_1 > 0$	·	T	T	·	···
$2(v_0 + k_0 + f) > n + 2t$	·	·	·	·	···
$2(v_1 + k_1 + f) > n + 2t$	·	·	T	·	···
$2(v_0 + k_0) > n + 2t$	·	·	·	·	···
$2(v_1 + k_1) > n + 2t$	·	·	·	·	···
$v_0 + k_0 + v_1 + k_1 + f \geq n$	T	T	T	·	···

Figure 2 An illustration of the successive abstractions.

The information of messages received by each process is used to check enabledness of transitions. However, the received messages necessarily form a subset of the sent messages. Using the notion of abstraction, this section proves that received messages can be forgotten without losing any information. Instead, it suffices to require the existence of a subset of sent messages that would enable a transition. Changing views from received messages to sent ones is often implicit [21, 20] and without restrictions it may introduce spurious counter-examples (see Example 13). By imposing that each message appears in at most one guard in the transitions taken by a process, the layering hypothesis guarantees that the abstraction is complete (Theorem 12). This abstraction is then used to provide a characterization of reachable configurations (Theorem 15), including those reachable via an infinite schedule.

A *succinct configuration* is an element of $C^s = P \to \overline{S^+}$. For $c^s \in C^s$, $p \in P$, $\ell < |c^s(p)|$ and $s \in S$, $c^s(p)(\ell) = s$ means that process p is/was in state s at layer ℓ. As before, if $\ell \geq |c^s(p)|$, then $c^s(p)(\ell) = \bot$, representing that process p has not reached layer ℓ yet. So-defined, the projection $\mathsf{state} : C^f \to C^s$ abstracts C^f into C^s, so that the reception actions become useless. The set of *succinct actions* is then $A^s = \bigcup_{\ell \in \mathbb{N}} \{[p : s \to s'] \mid p \in P, s \in S_\ell, s' \in S_{\ell+1}\}$ and the monoid morphism $\mathsf{simpl} : A^{f*} \to A^{s*}$ is defined by ignoring reception actions. Formally:

- for $\mathsf{rec}\,(p, p', \ell) \in A^f$, $\mathsf{simpl}\,(\mathsf{rec}\,(p, p', \ell)) = \varepsilon$;
- for $\mathsf{tr}\,(p, s, s') \in A^f$, $\mathsf{simpl}\,(\mathsf{tr}\,(p, s, s')) = [p : s \to s']$.

One can define enabledness of a succinct action, and its effect. For a succinct configuration $c^s \in C^s$ and a succinct action $[p : s \to s'] \in A^s$, writing $\ell = |c^s(p)| - 1$, then $[p : s \to s']$ is *enabled* at c^s if $\ell < \infty$, $c^s(p)(\ell) = s$ and $c^s(\_)(\ell) \in \uparrow \mathsf{guard}(s)(s')$. In this case, $([p : s \to s'](c^s))\,(p) = c^s(p) \cdot s'$ and $([p : s \to s'](c^s))$ coincides with c^s for any other process.

The first two conditions of enabledness are analogous to the case of the full semantics (see Definition 8). The last condition however replaces the guard of the edge with its upper closure. This derives from the fact that the condition now deals with *sent messages* instead of *received* ones, and the latter can only be smaller than the former.

Altogether, the *succinct semantics* of the LDTS consists of the POTS $\mathcal{O}^s_{\mathcal{D}} = (C^s, \sqsubseteq, A^s)$, whose definition is justified by the following proposition:

▶ **Proposition 10.** *The mappings* $\mathsf{state} : C^f \to C^s$ *and* $\mathsf{simpl} : A^{f*} \to A^{s*}$ *define an abstraction from the full POTS* $\mathcal{O}^f_{\mathcal{D}} = \left(C^f, \sqsubseteq, A^f\right)$ *to the succinct POTS* $\mathcal{O}^s_{\mathcal{D}} = (C^s, \sqsubseteq, A^s)$.

▶ **Example 11.** Consider the succinct configuration c^s in the top right of Figure 2. It is obtained by applying state to the full configuration c^f on the left. In Example 9, the full schedule $\sigma^f = \mathsf{rec}\,(p_0, p_2, 2) \cdot \mathsf{rec}\,(p_0, p_1, 2) \cdot \mathsf{tr}\,(p_0, v_1, v_1)$ is shown to be applicable at c^f. Therefore, Proposition 10 implies that $\mathsf{simpl}(\sigma^f) = [p_0 : v_1 \to v_1]$ is applicable at c^s.

Propositions 10 and 5 entail that the succinct abstraction is *sound* in the sense that it does not remove any existing behavior, and properties that hold on every execution of the succinct model also hold on the full semantics. However, in general, abstractions are not *complete* and they may introduce new behaviors (for instance, schedules without any reception actions may be applicable in the simplification but not in the full model). Nevertheless, the succinct abstraction is complete: there always exists an applicable full schedule corresponding to each applicable succinct schedule.

▶ **Theorem 12.** *Let* $\sigma^s \in \overline{A^{s*}}$ *be a succinct schedule applicable at an initial configuration* $c^s \in C^s$. *Then, there exists a full schedule* $\sigma^f \in \overline{A^{f*}}$ *applicable at a full configuration* $c^f \in C^f$ *such that:* $\mathsf{state}(c^f) = c^s$, $\mathsf{simpl}(\sigma^f) = \sigma^s$, *and* $\mathsf{state}(c^f \star \sigma^f) = c^s \star \sigma^s$.

To prove Theorem 12 one transforms each action $[p : s \to s']$ into a finite schedule of the form $\left(\mathsf{rec}\,(p, p_u, \ell)\right)_{u<U} \cdot \mathsf{tr}\,(p, s, s')$, carefully choosing the receptions to ensure that the last transition is enabled. To do so, the difficulties are twofold. First, the full schedule $\left(\mathsf{rec}\,(p, p_u, \ell)\right)_{u<U} \cdot \mathsf{tr}\,(p, s, s')$ not only depends on $[p : s \to s']$, but also on the current configuration. Therefore one cannot define a trivial abstraction. Second, this method requires a way to control the buffers of received messages throughout the schedule. Indeed, one should avoid that a process receives too many messages to take a transition, as 'un-receiving' messages in impossible. This is where the layered structure comes into play, and ensures that when a process receives messages enabling a transition, no earlier transition required these.

▶ **Example 13.** As explained, the layering assumption is crucial in Theorem 12. Consider the *non layered* distributed transition system with four states a, b, c, x, and two processes p, p'. Let c^f be the initial full configuration with $\mathsf{state}(c^f)(p) = a$ and $\mathsf{state}(c^f)(p') = x$. Intuitively, in this counterexample, the guards are set such that the first transition $\mathsf{tr}\,(p, a, b)$ is enabled only if $\mathsf{received}(c^f)(p)(p') = x$ while the next transition $\mathsf{tr}\,(p, b, c)$ requires $\mathsf{received}(c^f)(p)(p') = \bot \neq x$. Process p would thus have to "forget" that it received a message from p' in order to take the second transition, which is impossible in the full semantics.

In contrast, the succinct semantics does not record whether p has already received the message from p' when approaching the second transition. The succinct schedule $[p : a \to b] \cdot [p : b \to c]$ is therefore applicable at $\mathsf{state}(c^f)$ which would contradict Theorem 12 for unlayered distributed transition systems. Imposing that each message appears at most in one guard along the execution of a process, the layered hypothesis prevents this type of counterexamples.

The advantage of the succinct semantics over the full one is that the guards can only become true during an execution. This monotony property, combined with the layered hypothesis, entail the possibility to check that a configuration is reachable *a posteriori*, simply by verifying that the guards of the transitions that are taken are verified in the last configuration. In particular, this avoids building explicitly the schedule at all intermediate configurations. This is formally stated in the following definition and theorem.

▶ **Definition 14.** *A succinct configuration $c^s \in C^s$ is* coherent *if for any $p \in P$ and $\ell \in \mathbb{N}$, if $c^s(p)(\ell) = s \neq \bot$ and $c^s(p)(\ell+1) = s' \neq \bot$, then $c^s(\_)(\ell) \in \uparrow \mathsf{guard}(s, s')$.*

▶ **Theorem 15.** *Let $c^s, c^{s\prime} \in C^s$ be two succinct configurations such that c^s is* coherent*. Then the following statements are equivalent:*

- *$c^s \sqsubseteq c^{s\prime}$ and $c^{s\prime}$ is coherent.*
- *There exists a (possibly infinite) schedule $\sigma^s \in \overline{A^{s*}}$ applicable at c^s such that $c^s \star \sigma^s = c^{s\prime}$.*

3.3 Counter Abstraction

The theory presented so far dealt with a fixed set P of processes. As an advantage, the guards of the edges could be any condition on the set of received messages, but as a drawback, it is impossible to represent *parameterised* systems where the number of processes is not fixed. To remedy this downside, this section introduces *layered threshold automata* (LTA). While this model is syntactically similar to threshold automata [20], its semantics in terms of a POTS is novel. Natural abstractions between the semantics of LDTS and LTA can then be presented, proving that LTA form a faithful representation of distributed algorithms, in contrast to unrestricted threshold automata.

▶ **Definition 16.** *A* Layered Threshold Automaton *(LTA) is a tuple $\mathcal{T} = (R, S, \mathsf{guard})$ where:*

- *R is a set of* parameters
- *S is a set of* states *partitioned into layers: $S = \bigcup_{i=0}^{\infty} S_i$, with S_0 the set of* initial states*.*
- *$\mathsf{guard} : S^2 \to \mathsf{PA}(S \cup R)$ associates a* guard*, in Presburger arithmetic over free variables in $S \cup R$, to each pair of states. The* layered hypothesis *assumes that for $\ell \in \mathbb{N}$, $s \in S_\ell$, and $s' \in S$, $\mathsf{guard}(s, s') \in \mathsf{PA}(S_\ell \cup R)$ and if $s' \notin S_{\ell+1}$, $\mathsf{guard}(s, s') = \mathsf{false}$.*

The guards are monotonous*, i.e. for any guard $g \in \mathsf{guard}(S^2)$, for any valuation $\rho \in \mathbb{N}^R$, $\kappa, \kappa' \in \mathbb{N}^S$, if $\kappa \leq \kappa'$ when ordered pointwise and if $\rho, \kappa \models g$, then $\rho, \kappa' \models g$ as well.*

The set of parameters R typically includes the number n of processes and an upper bound t on the number of faulty processes. Intuitively, the guards represent the conditions on *sent messages* for taking the corresponding transition. The monotony assumption therefore requires that guards in the algorithms concern received messages only, which may be any subset of the sent messages.

In the remainder of this section, $\mathcal{T} = (R, S, \mathsf{guard})$ is a fixed LTA. A *configuration c* of $\mathcal{T}$ is defined by:

- a *parameter valuation* $\mathsf{param}(c) \in R \to \mathbb{N}$ that remains constant during an execution;
- a *counting mapping* $\kappa(c) \in S \to \mathbb{N}$ where $\kappa(c)(s) = k$ means that k processes have visited the state s;
- *flow counters* $\mathsf{flow}(c) \in \left(\bigcup_{\ell \in \mathbb{N}} S_\ell \times S_{\ell+1}\right) \to \mathbb{N}$ where $\mathsf{flow}(c)(s, s') = k$ means that k processes moved from s to s'.

Moreover, processes that leave a state must have entered it, therefore, configuations should also verify the following *flow conditions*:

- **in:** for every $\ell \in \mathbb{N} \setminus \{0\}$ and every $s \in S_\ell$, $\sum_{s' \in S_{\ell-1}} \mathsf{flow}(c)(s', s) = \kappa(c)(s)$
- **out:** for every $\ell \in \mathbb{N}$ and every $s \in S_\ell$, $\sum_{s' \in S_{\ell+1}} \mathsf{flow}(c)(s, s') \leq \kappa(c)(s)$.

The set C of all configurations is equipped with the natural order $\sqsubseteq$ defined by $c \sqsubseteq c'$ if $\mathsf{param}(c) = \mathsf{param}(c')$, $\kappa(c) \leq \kappa(c')$ and $\mathsf{flow}(c) \leq \mathsf{flow}(c')$.

An action over C is an element of $A = \bigcup_{\ell \in \mathbb{N}} A_\ell$ where for $\ell \in \mathbb{N}$, $A_\ell = \{[s \to s'] \mid s \in s_\ell, s' \in S_{\ell+1}\}$. For $c \in C$, an action $[s \to s'] \in A_\ell$ is *enabled* at c if:

- $\sum_{s'' \in S_{\ell+1}} \mathsf{flow}(c)(s, s'') < \kappa(c)(s)$, and
- $\mathsf{param}(c), \kappa(c) \models \mathsf{guard}(s, s')$, written $c \models \mathsf{guard}(s, s')$ for short.

In so, the successor configuration $[s \to s'](c) = c' \in C$ is defined by:

- $\mathsf{param}(c') = \mathsf{param}(c)$
- $\mathsf{flow}(c') = \mathsf{flow}(c) + \mathbb{1}_{(s,s')}$ where $\mathbb{1}_{(s,s')}(s, s') = 1$ and $\mathbb{1}_{(s,s')}(e) = 0$ elsewhere.
- $\kappa(c') = \kappa(c) + \mathbb{1}_{s'}$ where $\mathbb{1}_{s'}(s') = 1$ and $\mathbb{1}_{s'}(s'') = 0$ elsewhere.

One can easily check that configuration c' verifies the flow conditions.

The semantics of the LTA $\mathcal{T}$ is defined as the POTS $\mathcal{O}_\mathcal{T} = (C, \sqsubseteq, A)$.

For $\rho \in \mathbb{N}^R$, the set of configurations that have ρ as parameters and n processes initially is $C_\rho = \{c \in C \mid \mathsf{param}(c) = \rho, \text{ and } \sum_{s \in S_0} \kappa(c)(s) = \rho(\mathsf{n})\}$. Let $\mathcal{O}_\mathcal{T}^\rho = (C_\rho, \sqsubseteq, A)$ denote the POTS restricted to these configurations.

There is a strong link between LTA and LDTS. More precisely, fix a valuation $\rho \in \mathbb{N}^R$. Consider P_ρ a set of $\rho(\mathsf{n})$ processes, and the LDTS $\mathcal{D}_\rho = \left(P_\rho, S, \mathsf{guard}_\rho\right)$ where the function $\mathsf{guard}_\rho \in \bigcup_{\ell \in \mathbb{N}} \left(S_\ell \times S_{\ell+1} \to 2^{[P_\rho \to S_\ell^\perp]}\right)$ is defined for every $\ell \in \mathbb{N}$, $s \in S_\ell$ and $s' \in S_{\ell+1}$ by:

$$\mathsf{guard}_\rho(s, s') = \left\{x \in P \to S^\perp \mid \rho, \left[s \mapsto \left|x^{-1}(\{s\})\right|\right] \models \mathsf{guard}(s, s')\right\} \quad .$$

Let $C_\rho^s = P_\rho \to \overline{S^+}$ denote the set of succinct configurations of $\mathcal{D}_\rho$. Consider $c^s \in C_\rho^s$ and define $\mathsf{count}_{C_\rho^s}(c^s) \in C_\rho$ with:

- $\mathsf{param}\left(\mathsf{count}_{C_\rho^s}(c^s)\right) = \rho$
- for $\ell \in \mathbb{N}$ and $s \in S_\ell$: $\kappa\left(\mathsf{count}_{C_\rho^s}(c^s)\right)(s)(\ell) = |\{p \in P_\rho \mid c^s(p)(\ell) = s\}|$
- For $\ell \in \mathbb{N}$, $s \in S_\ell$ and $s' \in S_{\ell+1}$:

$$\mathsf{flow}\left(\mathsf{count}_{C_\rho^s}(c^s)\right)(s, s') = \left|\left\{p \in P_\rho \mid \begin{array}{r} c^s(p)(\ell) = s \\ c^s(p)(\ell+1) = s' \end{array}\right\}\right|$$

Let $A_\rho^s = \bigcup_{\ell \in \mathbb{N}} \{[p : s \to s'] \mid p \in P_\rho, s \in S_\ell, s' \in S_{\ell+1}\}$ denotes the set of succinct actions of $\mathcal{D}_\rho$. Define a monoid morphism $\mathsf{count}_{A_\rho^s} : A_\rho^{s*} \to A^*$ such that for $[p : s \to s'] \in A_\rho^s$, $\mathsf{count}_{A_\rho^s}(\mathsf{tr}\,(p, s, s')) = [s \to s']$. So defined:

▶ **Proposition 17.** *The mappings* $\mathsf{count}_{C_\rho^s} : C_\rho^s \to C_\rho$ *and* $\mathsf{count}_{A_\rho^s} : A_\rho^{s*} \to A^*$ *define an abstraction from the POTS* $\left(C_\rho^s, \sqsubseteq, A_\rho^s\right)$ *to the counter POTS* $(C_\rho, \sqsubseteq, A)$.

Proposition 17 holds for *any* parameter valuation $\rho \in \mathbb{N}^R$. Thus, a single LTA represents *infinitely-many* LDTS, one for each parameter valuation.

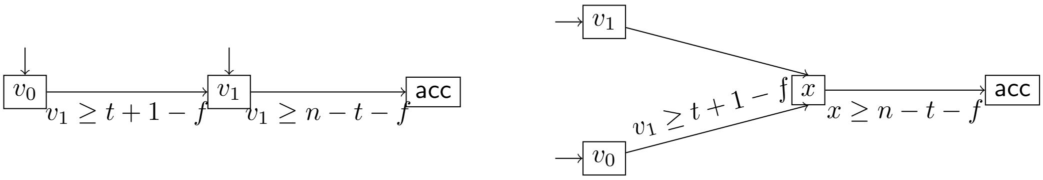
Similarly to the case of LTA, one can define *coherence* of configurations for LDTS, and obtain an equivalent of Theorem 15 at the counter abstraction level.

▶ **Definition 18.** *Configuration* $c \in C$ *is said* counter coherent *when for every* $\ell \in \mathbb{N}$, $s \in S_\ell$ *and* $s' \in S_{\ell+1}$, *if* $\mathsf{flow}(c)(s, s') > 0$, *then* $c \models \mathsf{guard}(s, s')$.

▶ **Theorem 19.** *Let* $c, c' \in C_\rho$ *be two configurations such that* c *is counter coherent. Then the following statements are equivalent:*

- $c \sqsubseteq c'$ *and* c' *is counter coherent;*
- *There exists a (possibly infinite) schedule* $\sigma \in \overline{A^*}$ *applicable at* c *such that* $c \star \sigma = c'$.

The flow conditions and the counter coherence can easily be encoded as a set of linear arithmetic formulas that *do not* depend on the number of processes. In particular, if the LTA is *finite*, then the resulting set of equations is finite as well, making the reachability problem decidable in this case (for initial and target states represented by linear arithmetic formulas). This can be used to verify not only safety properties, but also liveness properties as configurations represent potentially infinite behaviors and contain information about the whole execution. Theorem 19 differs from the threshold automata approach [20] because a schedule does not need to be explicitely built. In particular, the layering assumption implies that the order in which guards become true is irrelevant, which simplifies a lot the SMT queries. More importantly, our approach applies to *infinite* automata where methods based on bounding the diameter of the transition system have little chance of succeeding.



(a) Non-layered.

(b) Layered.

Figure 3 Two threshold automata for the reliable broadcast algorithm [11].

▶ **Example 20.** Theorem 19 heavily relies on the layered hypothesis. To see that, consider the non layered model of Figure 3a. Let c be a configuration with $\mathsf{flow}(c)(v_0, v_1) > 0$. Then the counter coherence would require that $c \models v_1 \geq t{+}1{-}f$, however, this last condition may only hold because the transition was taken in the first place, resulting in spurious configurations. This can be fixed by tweaking the model in order to make it layered as seen on Figure 3b.

3.4 Guard Abstraction

Consider an LTA $\mathcal{T} = (R, S, \mathsf{guard})$. Even when S is finite, its configuration set C is infinite as the number of processes n is unbounded. When S is infinite, then C is infinite in two dimensions: it consists of infinitely many variables that may take infinitely many values. The guard abstraction presented here aims at partitioning these values into finitely many classes. The resulting model will however remain infinite, if S is.

Consider a set $G \subset \mathsf{PA}(S \cup R)$ of *monotonous guards*, that is, every $g \in G$ is a linear arithmetic formulas with free variables in $S \cup R$ such that for $\rho \in \mathbb{N}^R$ and $\kappa, \kappa' \in \mathbb{N}^S$, if $\kappa \leq \kappa'$ pointwise and if $\rho, \kappa \models g$, then $\rho, \kappa' \models g$ as well.

Intuitively, the guard abstraction only records the valuations of the guards, not the number of processes in each state. For this idea to succeed, the valuations of the guards must converge during an execution, which is guaranteed by the following proposition.

▶ **Proposition 21.** *The mapping* $\mathsf{eval}_G \colon (C, \sqsubseteq) \to (2^G, \subseteq)$ *defined by* $\mathsf{eval}_G(c) = \{g \in G \mid c \models g\}$ *is Scott-continuous.*

4 Guard Automata towards Practical Implementation

While Theorem 19 suffices to verify *finite* LTA through the counter abstraction, it falls short at capturing infinite models that arise for instance from round-based algorithms. This section introduces guard automata as a finite-state abstraction which is sound, yet, unsurprisingly, not complete in general and may introduce spurious counterexamples.

4.1 Cyclic LTA

Towards algorithmic considerations and practical implementations, the rest of the paper focuses on round-based distributed algorithms, which can be captured by cyclic LTA. Intuitively, a cyclic LTA is used to model an LTA that repeats a finite series of layers indefinitely. For $k \in \mathbb{N}_{>0}$, a *k-cyclic LTA* (k-CLTA) is a tuple $\mathcal{T}^c = (R, S^c, \mathsf{guard}^c)$ where:

- R is a *finite* set of parameters.
- S^c is a *finite* set of states partitioned into k layers $S^c = S^c_0 \cup \cdots \cup S^c_{k-1}$.
- $\mathsf{guard}^c : S^{c2} \to \mathsf{PA}(R \cup S^c)$ is a finite set of guards such that for $\ell < k$, $s^c \in S^c_\ell$ and $s^{c\prime} \in S^c$, $\mathsf{guard}^c(s^c, s^{c\prime}) \in \mathsf{PA}(R \cup S^c_\ell)$ and if $s^{c\prime} \notin S^c_{\ell+1 \bmod k}$, then $\mathsf{guard}^c(s^c, s^{c\prime}) = \mathsf{false}$.

Unfolding a k-CLTA yields an infinite-state acyclic LTA $\mathsf{unfold}\,(R, S^c, \mathsf{guard}^c)$. Formally $\mathsf{unfold}\,(R, S^c, \mathsf{guard}^c) = (R, S, \mathsf{guard})$ with:

- $S = \{(s^c, \ell) \mid \ell \in \mathbb{N}, s^c \in S^c_{\ell \bmod k}\}$
- For $\ell \in \mathbb{N}$, $s^c \in S^c_{\ell \bmod k}$ and $s^{c\prime} \in S^c_{\ell+1 \bmod k}$, $\mathsf{guard}\left((s^c, \ell), (s^{c\prime}, \ell+1)\right) = \mathsf{guard}^c(s^c, s^{c\prime})[s^{c\prime\prime} \leftarrow (s^{c\prime\prime}, \ell) \text{ for } s^{c\prime\prime} \in S^c_{\ell \bmod k}]$ meaning that any free variable $s^{c\prime\prime} \in S^c$ that appears in $\mathsf{guard}^c(s^c, s^{c\prime})$ gets replaced with $(s^{c\prime\prime}, \ell)$. In any other case, guard is false.

4.2 Guard Automaton

From the guard abstraction, one can construct a finite-state automaton that represents the set of reachable configurations of a cyclic LTA.

Let $\mathcal{T}^c = (R, S^c, \mathsf{guard}^c)$ be a k-CLTA equipped with a *finite* set of guards expressed in Presburger arithmetic: $G^c = \bigcup_{\ell<k} G^c_\ell$ such that for $\ell < k$, $G^c_\ell \in \mathsf{PA}(S^c_\ell \cup R)$. In practice, G^c will include all guards appearing in the LTA, as well as the events that need to be observed.

A CLTA can be unfolded into an infinite-state LTA, by concatenating copies of $\mathcal{T}^c$. In order for the guard abstraction to be formally defined, copies of the guards in G^c for each new layer are required. For $\ell \in \mathbb{N}$ a layer index and $g^c \in G^c_{\ell \bmod k}$ a guard, $\mathsf{unfold}^G{}_\ell(g^c) = g^c[s^c \leftarrow (s^c, \ell) \text{ for } s^c \in S^c_{\ell \bmod k}]$ denotes the guard obtained by replacing every free occurrence of a variable $s^c \in S^c_{\ell \bmod k}$ in g^c by (s^c, ℓ). The converse folding operation is defined by: $\mathsf{fold}^G{}_\ell(g) = g[(s^c, \ell) \leftarrow s^c, \text{ for } s^c \in S^c_{\ell \bmod k}]$. Finally, $G_\ell = \mathsf{unfold}^G\,(G^c_{\ell \bmod k})$ is the set of guards at layer ℓ and $G = \bigcup_{\ell \in \mathbb{N}} G_\ell$ the set of all guards.

The guard abstraction maps every configuration of $\mathsf{unfold}(\mathcal{T}^c)$ to a set of guards that hold in that configuration. Formally, $\mathsf{eval}_G : C \to 2^G$. A set of guards $\gamma \in 2^G$ can be represented with the sequence $\gamma_0\gamma_1 \ldots$, where for $\ell \in \mathbb{N}$, $\gamma_\ell = \gamma \cap G_\ell$. $\mathsf{fold}^G(\gamma)$ then denotes the sequence $\mathsf{fold}^G{}_0(\gamma_0) \cdot \mathsf{fold}^G{}_1(\gamma_1) \cdots \in \left(2^{G^c}\right)^\omega$ and unfold^G is the converse operation that applies $\mathsf{unfold}^G{}_\ell$ to the elements of layer ℓ in the sequence. Doing so, a configuration $c \in C$ defines a (possibly infinite) word $\gamma^c_0\gamma^c_1 \ldots$ over the *finite* alphabet $\Sigma = \bigcup_{\ell<k} 2^{G^c_\ell}$ as represented in Figure 4.

$$\begin{tikzcd}
c \in C \arrow[r, "\mathsf{eval}_G : C \to 2^G \approx \prod_{\ell \in \mathbb{N}} 2^{G_\ell}"] & \begin{array}{c} \gamma_0 \subset G_0 \\ \gamma_1 \subset G_1 \\ \gamma_2 \subset G_2 \\ \vdots \\ \gamma_{k-1} \subset G_{k-1} \\ \gamma_k \subset G_k \\ \gamma_{k+1} \subset G_{k+1} \\ \vdots \end{array} \arrow[r, shift left, "\mathsf{fold}^G : \prod_{\ell \in \mathbb{N}} 2^{G_\ell} \to \prod_{\ell \in \mathbb{N}} 2^{G^c_{\ell \bmod k}}"] & \begin{array}{c} \gamma^c_0 \subset G^c_0 \\ \gamma^c_1 \subset G^c_1 \\ \gamma^c_2 \subset G^c_2 \\ \vdots \\ \gamma^c_{k-1} \subset G^c_{k-1} \\ \gamma^c_k \subset G^c_0 \\ \gamma^c_{k+1} \subset G^c_1 \\ \vdots \end{array} \arrow[l, shift left, "\mathsf{unfold}^G : \prod_{\ell \in \mathbb{N}} 2^{G^c_{\ell \bmod k}} \to \prod_{\ell \in \mathbb{N}} 2^{G_\ell}"]
\end{tikzcd}$$

Figure 4 From a configuration to a word over the finite alphabet of the guard automaton.

For $\ell < k$ a layer index, $\gamma^c \in 2^{G^c_\ell}$ and $\gamma^{c\prime} \in 2^{G^c_{\ell+1 \bmod k}}$ guard valuations of layer ℓ and the next layer, one can use an SMT solver to check whether $\gamma^{c\prime}$ is a successor γ^c. Precisely, the SMT query asks for the existence of $x \in \mathbb{N}^{S^c_\ell}$, $y \in \mathbb{N}^{S^c_{\ell+1 \bmod k}}$ and $e \in \mathbb{N}^{S^c_\ell \times S^c_{\ell+1 \bmod \mathbb{N}}}$ such that the valuation of guards (1), flow condition (2) and counter coherence (3) are verified.

$$x \models \bigwedge_{g^c \in \gamma^c} g^c \wedge \bigwedge_{g^c \in G^c_\ell \setminus \gamma^c} \neg g^c \qquad y \models \bigwedge_{g^{c\prime} \in \gamma^{c\prime}} g^{c\prime} \wedge \bigwedge_{g^{c\prime} \in G^c_{\ell+1 \bmod k} \setminus \gamma^{c\prime}} \neg g^{c\prime} \tag{1}$$

$$e, x \models \bigwedge_{s^c \in S^c_\ell} s^c \geq \sum_{s^{c\prime} \in S^c_{\ell+1 \bmod k}} \left[s^c, s^{c\prime}\right] \qquad e, y \models \bigwedge_{s^{c\prime} \in S^c_{\ell+1 \bmod k}} \sum_{s^c \in S^c_\ell} \left[s^c, s^{c\prime}\right] = s^{c\prime} \tag{2}$$

$$e, x \models \bigwedge_{(s^c, s^{c\prime}) \in S^c_\ell \times S^c_{\ell+1 \bmod \mathbb{N}}} \left[s^c, s^{c\prime}\right] > 0 \longrightarrow \mathsf{guard}^c(s^c, s^{c\prime}) \tag{3}$$

The guard automaton is a finite automaton whose language *overapproximates* the set of reachable configurations. It bears similarities with de Bruijn graphs [15] used e.g. in bioinformatics. If $E_\ell \subset 2^{G^c_\ell} \times 2^{G^c_{\ell+1 \bmod k}}$ denotes the set of all pairs $\gamma^c, \gamma^{c\prime}$ that verify conditions (1) and and (3), one can build the set $E = \bigcup_{\ell < k} E_\ell$.

▶ **Definition 22.** *The* guard automaton *of* $\mathcal{T}^c$ *is* $\mathsf{GA}_G(\mathcal{T}^c) = \left(\Sigma, E, 2^{G^c_0}, \mathsf{src}, \mathsf{dest}, \mathsf{label}\right)$ *where:*

- Σ *is both the alphabet and the set of states.*
- $2^{G^c_0} \subset \Sigma$ *is the set of initial states.*
- $E \subset \Sigma^2$ *defined above is the set of edges, equipped with* $\mathsf{src} : E \to \Sigma$ *(resp.* $\mathsf{dest} : E \to \Sigma$*) that defines the* source state *(resp.* destination state*) of every edge, and* $\mathsf{label} : E \to \Sigma$ *associates a* label *to each edge defined by* $\mathsf{label}(\gamma^c, \gamma^{c\prime}) = \gamma^c$*.*

An infinite run $(e_\ell)_{\ell<\infty}$ of the guard automaton defines a word $\mathsf{word}\left((e_\ell)_{\ell<\infty}\right) = \mathsf{label}(e_0) \cdot \mathsf{label}(e_1) \cdot \cdots$, and $\mathcal{L}(\mathsf{GA}_G(\mathcal{T}^c)) \subset \Sigma^\omega$ denotes the language of $\mathsf{GA}_G(\mathcal{T}^c)$.

▶ **Example 23.** Algorithm 1 can be described by the following CLTA with $k = 1$. The parameters are $R = \{n, t, f\}$ where f denotes the actual number of Byzantine faults. States are $S^c = \{v_0, k_0, v_1, k_1\}$. The guards here only depend on the next value of v. For instance:

$$\mathsf{guard}(\_, v_0) = (v_0 + k_0 + v_1 + k_1 + f = n)$$
$$\wedge \Big(\big(2(v_0 + k_0 + f) > n + 2t\big) \vee \big((2v_0 + 2k_0 \leq n + 2t) \wedge (2v_1 + 2k_1 \leq n + 2t) \wedge (k_1 = 0)\big)\Big).$$

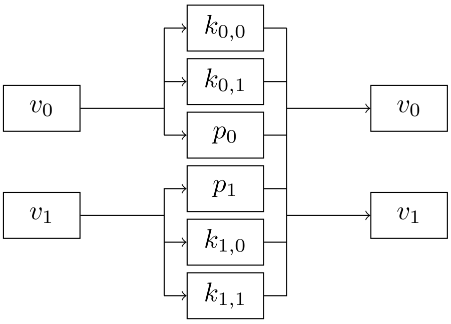
Also, $\mathsf{guard}(\_, k_0) = \mathsf{guard}(\_, v_0)$ and $\mathsf{guard}(\_, v_1) = \mathsf{guard}(\_, k_1)$ is defined symmetrically.

A configuration c of the unfolded LTA is depicted bottom-right of Figure 2, where the array contains the valuation $\kappa(c)$ and the arrows represent the flow. For example $\kappa(c)(v_1, 0) = 2$, $\mathsf{flow}(c)((v_0, 0), (k_1, 1)) = 1$ and $\mathsf{flow}(c)((v_0, 0), (v_0, 1)) = 0$.

The guard abstraction transforms c into the guard configuration bottom-left of Figure 2. Here, we chose the set of guards G^c to consist of $s > 0$ for each $s \in S^c$ and of the guards of the LTA. The alphabet Σ contains e.g., $(T \cdot T \cdot \cdots \cdot T)$. SMT queries determine whether two letters may appear successively, in order to build the guard automaton. For instance, according to the first two layers of $\mathsf{eval}_G(c)$, $(T \cdot T \cdot \cdots \cdot T)$ can be followed by $(T \cdot TT \cdot \cdots \cdot T)$. There will therefore be a transition between these two states in the guard automaton.

▶ **Theorem 24.** *Let* $c \in C$ *be a configuration of* $\mathsf{unfold}(\mathcal{T}^c)$ *and* $\mathsf{eval}_G(c) \in 2^G$ *its guard abstraction. If c is counter-coherent, then* $\mathsf{fold}^G\left(\mathsf{eval}_G(c)\right) \in \mathcal{L}(\mathsf{GA}_G(\mathcal{T}^c))$*.*

By soundness of the guard automaton construction, a property which holds on configurations that correspond to runs of $\mathsf{GA}_G(\mathcal{T}^c)$ also holds on the configurations of $\mathsf{unfold}(\mathcal{T}^c)$. A simple verification procedure thus consists in checking that $\mathcal{L}(\mathsf{GA}_G(\mathcal{T}^c))$ is included in a given language of correct configurations. At a first glance, it might seem that only safety properties can be checked. However, the guard automaton also represents configurations reachable by infinite schedules, making the verification of liveness properties feasible.



$P = \{n, t, f\}$ and for $x, y \in \{0, 1\}$:

$$\begin{aligned}
\mathsf{guard}(v_x, p_x) &= \mathsf{true}\\
\mathsf{guard}(v_x, k_{x,y}) &= [2(v_y + f) \geq n]\\
\mathsf{guard}(\_, v_x) &= [2(p_x + k_{x,0} + k_{x,1}) > n + 2t] \vee\\
&\quad \begin{bmatrix} 2(p_0 + k_{0,0} + k_{0,1}) \leq n + 2t \wedge \\ 2(p_1 + k_{1,0} + k_{1,1}) \leq n + 2t \wedge \\ k_{0,x} + k_{1,x} = 0 \end{bmatrix}
\end{aligned}$$

Figure 5 A 2-CLTA for the Phase King algorithm with non-deterministic choice of the king. A process in $k_{x,y}$ is king of the current round, its current value is x and it thinks the majority is y.

▶ **Example 25.** For presentation purposes, Algorithm 1 is an overly simplified version of the Phase King algorithm [8]. The latter can be faithfully encoded by the 2-CLTA $\mathcal{T}^c$ of Figure 5, where the updated value when there is no clear majority is not the king's value, but rather the majority of the values received by the king. Each round consists of two layers of communication, a first in which each process broadcasts its value, and a second in which the king broadcasts what it thinks is the majority. The set of guards at the first layer is $G_0^c = \{v_0 > 0, v_1 > 0\}$ and at the second layer G_1^c consists of $k_{0,0}+k_{1,0} > 0$, $k_{0,1}+k_{1,1} > 0$, $p_0+k_{0,0}+k_{0,1} > 0$, $p_1+k_{1,0}+k_{1,1} > 0$, $2(k_{0,0}+k_{0,1}+p_0+f) > n+2t$ and $2(k_{1,0}+k_{1,1}+p_1+f) > n+2t$.

Restricting to valuations with $\sum_{s \in S_\ell} s+f = n$ (fairness) and $k_{0,0}+k_{0,1}+k_{1,0}+k_{1,1} \leq 1$ (at most one king), the resulting guard automaton has 3 states in even layers and 11 in odd layers. Writing [*formula*] for the set of letters in 2^{G^c} for which *formula* holds, one can show:

$$\begin{aligned}
\mathcal{L}(\mathsf{GA}_G(\mathcal{T}^c)) \subset\ & [\neg(k_{0,0} + k_{1,0} > 0) \wedge \neg(k_{0,1} + k_{1,1} > 0)]^\omega && (4)\\
& \cup \Sigma^*[(k_{0,0} + k_{1,0} > 0) \vee (k_{0,1} + k_{1,1} > 0)][\neg(p_0 + k_{0,0} + k_{0,1} > 0)]^\omega && (5)\\
& \cup \Sigma^*[(k_{0,0} + k_{1,0} > 0) \vee (k_{0,1} + k_{1,1} > 0)][\neg(p_1 + k_{1,0} + k_{1,1} > 0)]^\omega\ . && (6)
\end{aligned}$$

Therefore, either every chosen king is Byzantine (4), or all processes agree on a value after a non-Byzantine king is chosen (5 or 6).

In general, although is it sound, the guard automaton construction is not complete: the language may contain words that correspond to no configuration of the LTA. As usual for incomplete methods, heuristics can be used to remove some spurious counterexamples.

5 Conclusion

This paper presented a methodology, based on domain theory, to represent and analyze distributed algorithms. Infinite-state models are abstracted into finite-state guard automata, on which one can check safety and liveness properties.

Optimizing and benchmarking the guard automaton implementation is on our current agenda to demonstrate the applicability of our methodology to standard distributed algorithms. A more long-term research objective is to build on the current contribution to develop a rigorous framework for the verification of randomized distributed algorithms.

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Zeinab Ganjei, Ahmed Rezine, and Yunyun Zhu. Verification of cache coherence protocols wrt. trace filters. In *Proceedings of the 15th International Conference on Formal Methods in Computer-Aided Design (FMCAD'15)*, pages 9–16. IEEE, 2015.

2 Samson Abramsky and Achim Jung. *Domain Theory*, volume 3 of *Handbook of Logic in Computer Science*, pages 1–168. Clarendon Press, 1994.

3 C. Aiswarya, Benedikt Bollig, and Paul Gastin. An automata-theoretic approach to the verification of distributed algorithms. *Information and Computation*, 259:305–327, 2018.

4 Francesco Alberti, Silvio Ghilardi, and Elena Pagani. Counting constraints in flat array fragments. In *Proceedings of the 8th International Joint Conference on Automated Reasoning (IJCAR'16)*, volume 9706 of *Lecture Notes in Computer Science*, pages 65–81, 2016.

5 A. R. Balasubramanian, Javier Esparza, and Marijana Lazić. Complexity of verification and synthesis of threshold automata. In *Proceedings of the 18th International Symposium on Automated Technology for Verification and Analysis (ATVA'20)*, volume 12302 of *Lecture Notes in Computer Science*, pages 144–160. Springer, 2020.

6 Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *Proceedings of the 2nd Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC'83)*, pages 27–30, 1983.

7 Idan Berkovits, Marijana Lazic, Giuliano Losa, Oded Padon, and Sharon Shoham. Verification of threshold-based distributed algorithms by decomposition to decidable logics. In *Proceedings of the 31st International Conference on Computer Aided Verification (CAV'19)*, volume 11562 of *Lecture Notes in Computer Science*, pages 245–266. Springer, 2019.

8 Piotr Berman and Juan A. Garay. Cloture votes: n/4-resilient distributed consensus in t+1 rounds. *Mathematical Systems Theory*, 26(1):3–19, 1993.

9 Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. *Decidability of Parameterized Verification.* Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2015.

10 Luca Bortolussi and Jane Hillston. Model checking single agent behaviours by fluid approximation. *Information and Computation*, 242:183–226, 2015.

11 Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840, 1985.

12 Mouna Chaouch-Saad, Bernadette Charron-Bost, and Stephan Merz. A reduction theorem for the verification of round-based distributed algorithms. In *Proceedings of the 3rd International Workshop on Reachability Problems (RP'09)*, volume 5797 of *Lecture Notes in Computer Science*, pages 93–106, 2009.

13 Bernadette Charron-Bost and André Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.

14 Andrei Damian, Cezara Drăgoi, Alexandru Militaru, and Josef Widder. Communication-closed asynchronous protocols. In *Proceedings of the 31st International Conference on Computer Aided Verification (CAV'19)*, volume 11562 of *Lecture Notes in Computer Science*, pages 344–363. Springer, 2019.

15 Nicolaas G. de Bruijn. A combinatorial problem. *Indagationes Mathematicae*, 49:758–764, 1946.

16 Cezara Drăgoi, Thomas A. Henzinger, Helmut Veith, Josef Widder, and Damien Zufferey. A Logic-Based Framework for Verifying Consensus Algorithms. In *Proceedings of the 15th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'14)*, volume 8318 of *Lecture Notes in Computer Science*, pages 161–181, 2014.

17 Tzilla Elrad and Nissim Francez. Decomposition of distributed programs into communication-closed layers. *Science of Computer Programming*, 2(3):155–173, 1982.

18 Javier Esparza. Keeping a crowd safe: On the complexity of parameterized verification (invited talk). In *Proceedings of the 31st International Symposium on Theoretical Aspects of Computer Science (STACS'14)*, volume 25 of *LIPIcs*, pages 1–10. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2014.

19 Igor Konnov, Marijana Lazić, Helmut Veith, and Josef Widder. A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL'17)*, pages 719–734, 2017.

20 Igor Konnov, Helmut Veith, and Josef Widder. On the completeness of bounded model checking for threshold-based distributed algorithms: Reachability. *Information and Computation*, 252:95–109, 2017.

21 Jure Kukovec, Igor Konnov, and Josef Widder. Reachability in parameterized systems: All flavors of threshold automata. In *Proceedings of the 29th International Conference on Concurrency Theory (CONCUR'18)*, volume 118 of *LIPIcs*, pages 19:1–19:17, 2018.

22 Leslie Lamport. Checking a multithreaded algorithm with $^{+}$CAL. In *Proceedings of the 20th International Symposium on Distributed Computing (DISC'06)*, volume 4167 of *Lecture Notes in Computer Science*, pages 151–163. Springer, 2006.

23 Richard J. Lipton. Reduction: A method of proving properties of parallel programs. *Communications of the ACM*, 18(12):717–721, 1975.

24 Ognjen Maric, Christoph Sprenger, and David A. Basin. Cutoff bounds for consensus algorithms. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV'17)*, volume 10427 of *Lecture Notes in Computer Science*, pages 217–237, 2017.

25 Kenneth L. McMillan. Parameterized verification of the FLASH cache coherence protocol by compositional model checking. In *Proceedings of the 11th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'01)*, volume 2144, pages 179–195. Springer, 2001.

26 Dominique Méry. Verification by Construction of Distributed Algorithms. In *Proceedings of the 16th International Colloquium on Theoretical Aspects of Computing (ICTAC'19)*, volume 11884 of *Lecture Notes in Computer Science*, pages 22–38. Springer, 2019.

27 Arnaud Sangnier, Nathalie Sznajder, Maria Potop-Butucaru, and Sébastien Tixeuil. Parameterized verification of algorithms for oblivious robots on a ring. In *Proceedings of the 17th International Conference on Formal Methods in Computer Aided Design (FMCAD'17)*, pages 212–219. IEEE, 2017.

28 Ocan Sankur and Jean-Pierre Talpin. An abstraction technique for parameterized model checking of leader election protocols: Application to FTSP. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'17)*, volume 10205 of *Lecture Notes in Computer Science*, pages 23–40, 2017. doi:10.1007/978-3-662-54577-5_2.

29 Ilina Stoilkovska, Igor Konnov, Josef Widder, and Florian Zuleger. Verifying safety of synchronous fault-tolerant algorithms by bounded model checking. In *Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'19)*, volume 11428 of *Lecture Notes in Computer Science*, pages 357–374, 2019.

Dynamic Data-Race Detection Through the Fine-Grained Lens

Rucha Kulkarni
University of Illinois at Urbana-Champaign, IL, USA

Umang Mathur
University of Illinois at Urbana-Champaign, IL, USA

Andreas Pavlogiannis
Aarhus University, Denmark

Abstract

Data races are among the most common bugs in concurrency. The standard approach to data-race detection is via dynamic analyses, which work over executions of concurrent programs, instead of the program source code. The rich literature on the topic has created various notions of dynamic data races, which are known to be detected efficiently when certain parameters (e.g., number of threads) are small. However, the *fine-grained* complexity of all these notions of races has remained elusive, making it impossible to characterize their trade-offs between precision and efficiency.

In this work we establish several fine-grained separations between many popular notions of dynamic data races. The input is an execution trace σ with $\mathcal{N}$ events, $\mathcal{T}$ threads and $\mathcal{L}$ locks. Our main results are as follows. First, we show that *happens-before* HB *races* can be detected in $O(\mathcal{N} \cdot \min(\mathcal{T}, \mathcal{L}))$ time, improving over the standard $O(\mathcal{N} \cdot \mathcal{T})$ bound when $\mathcal{L} = o(\mathcal{T})$. Moreover, we show that even reporting an HB race that involves a read access is hard for 2-orthogonal vectors (2-OV). This is the first rigorous proof of the conjectured quadratic lower-bound in detecting HB races. Second, we show that the recently introduced *synchronization-preserving races* are hard to detect for 3-OV and thus have a cubic lower bound, when $\mathcal{T} = \Omega(\mathcal{N})$. This establishes a complexity separation from HB races which are known to be strictly less expressive. Third, we show that *lock-cover races* are hard for 2-OV, and thus have a quadratic lower-bound, even when $\mathcal{T} = 2$ and $\mathcal{L} = \omega(\log \mathcal{N})$. The similar notion of *lock-set races* is known to be detectable in $O(\mathcal{N} \cdot \mathcal{L})$ time, and thus we achieve a complexity separation between the two. Moreover, we show that lock-set races become hitting-set (HS)-hard when $\mathcal{L} = \Theta(\mathcal{N})$, and thus also have a quadratic lower bound, when the input is sufficiently complex. To our knowledge, this is the first work that characterizes the complexity of well-established dynamic race-detection techniques, allowing for a rigorous comparison between them.

2012 ACM Subject Classification Software and its engineering → Software testing and debugging; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases dynamic analyses, data races, fine-grained complexity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.16

Related Version *Full Version*: `https://arxiv.org/abs/2107.03569`

Funding *Umang Mathur*: partially funded by a Google PhD Fellowship and by the Simons Institute for the Theory of Computing.

1 Introduction

Concurrent programs that communicate over shared memory are prone to *data races*. Two events are *conflicting* if they access the same memory location and one (at least) modifies that location. Data races occur when conflicting accesses happen concurrently between different threads, and form one of the most common bugs in concurrency. In particular, data races are often symptomatic of bugs in software like data corruption [5, 20, 27], and they have been deemed *pure evil* [6] due to the problems they have caused in the past [44]. Moreover, many compiler optimizations are unsound in the presence of data races [37, 41], while data-race freeness is often a requirement for assigning well-defined semantics to programs [7].

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 16; pp. 16:1–16:23
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The importance of data races in concurrency has led to a multitude of techniques for detecting them efficiently [4, 40]. By far the most standard approach is via *dynamic analyses*. Instead of analyzing the full program, dynamic analyzers try to *predict* the existence of data races by observing and analyzing concurrent executions [38, 21, 29]. As full dynamic data race prediction is NP-hard in general [25], researchers have developed several approximate notions of dynamic races, accompanied by efficient techniques for detecting each notion.

Happens-before races. The most common technique for detecting data races dynamically is based on Lamport's *happens-before (HB)* partial order [23]. Two conflicting events form an HB race if they are unordered by HB, as the lack of ordering between them indicates the fact that they may execute concurrently, thereby forming a data race. The standard approach to HB race detection is via the use of vector clocks [19], and has seen wide success in commercial race detectors [36]. As vector clock computation is known to require $\Theta(\mathcal{N} \cdot \mathcal{T})$ time on traces of $\mathcal{N}$ events and $\mathcal{T}$ threads [10], HB race detection is often assumed to suffer the same bound, and has thus been a subject of further practical optimizations [30, 16].

Synchronization preserving races. HB races were recently generalized to sync(hronization)-preserving races [26]. Intuitively, two conflicting events are in a sync-preserving race if the observed trace can be soundly reordered to a witness trace in which the two events are concurrent, but without reordering synchronization events (e.g., locking events). Similar to HB races, sync-preserving races can be detected in linear time when the number of threads is constant. However, the dependence on the number of threads is cubic for sync-preserving races, as opposed to the linear dependence for HB races. On the other hand, sync-preserving races are known to offer better precision in program analysis.

Races based on the locking discipline. The locking discipline dictates that threads that access a common memory location must do so inside *critical sections*, using a common lock, when performing the access [40]. Although this discipline is typically not enforced, it is considered good practice, and hence instances that violate this principle are often considered indicators of erroneous behavior. For this reason, there have been two popular notions of data races based on the locking discipline, namely *lock-cover races* [14] and *lock-set races* [34]. Both notions are detectable in linear time when the number of locks is constant, however, lock-set race detection is typically faster in practice, which also comes at the cost of being less precise.

Observe that, although techniques for all aforementioned notions of races are generally thought to operate in linear time, they only do so assuming certain parameters, such as the number of threads, are constant. However, as these techniques are deployed in runtime, often with extremely long execution traces, they have to be as efficient as absolutely possible, often in scenarios when these parameters are very large. When a data-race detection technique is too slow for a given application, the developers face a dilemma: do they look for a faster algorithm, or for a simpler abstraction (i.e., a different notion of dynamic races)? For these reasons, it is important to understand the *fine-grained* complexity of the problem at hand with respect to such parameters. Fine-grained lower bounds can rule out the possibility of faster algorithms, and thus help the developers focus on new abstractions that are more tractable for the given application. Motivated by such questions, in this work we settle the fine-grained complexity of dynamically detecting several popular notions of data races.

1.1 Our Contributions

Here we give a full account of the main results of this work, while we refer to later sections for precise definitions and proofs. We also refer to Section 2.3 for relevant notions in fine-grained complexity and popular hypotheses. The input is always a concurrent trace σ of length $\mathcal{N}$, consisting of $\mathcal{T}$ threads, $\mathcal{L}$ locks, and $\mathcal{V}$ variables.

Happens-before race. We first study the fine-grained complexity of HB races, as they form the most popular class of dynamic data races. The task of most techniques is to report all events in σ that participate in an HB race, which is known to take $O(\mathcal{N} \cdot \mathcal{T})$ time [19]. Note that the bound is quadratic when $\mathcal{T} = \Theta(\mathcal{N})$, and multiple heuristics have been developed to address it in practice (see e.g., [16]). Our first result shows that polynomial improvements below this quadratic bound are unlikely.

▶ **Theorem 1.** *For any $\epsilon > 0$, there is no algorithm that detects even a single HB race that involves a read in time $O(\mathcal{N}^{2-\epsilon})$, unless the OV hypothesis fails.*

Orthogonal vectors (OV) is a well-studied problem with a long-standing quadratic worst-case upper bound. The associated hypothesis states that there is no sub-quadratic algorithm for the problem [43]. It is also known that the strong exponential time hypothesis (SETH) implies the Orthogonal Vectors hypothesis [42]. Thus, under the OV hypothesis, Theorem 1 establishes a quadratic lower bound for HB race detection.

Note that the hardness of Theorem 1 arises out of the requirement to detect HB races that involve a read. A natural follow-up question is whether detecting if the input contains *any* HB race (i.e., not necessarily involving a read) has a similar lower bound based on SETH. Our next theorem shows that under the non-deterministic SETH (NSETH) [9], there is no fine-grained reduction from SETH that proves any lower bound for this problem above $\mathcal{N}^{3/2}$.

▶ **Theorem 2.** *For any $\epsilon > 0$, there is no $(2^{\mathcal{N}}, \mathcal{N}^{3/2+\epsilon})$-fine-grained reduction from SAT to the problem of detecting any HB race, unless NSETH fails.*

Given the impossibility of Theorem 2, it would be desirable to at least show a super-linear lower bound for detecting any HB data race. To tackle this question, we show that detecting any HB race is hard for the general problem of model checking first-order formulas quantified by $\forall\exists\exists$ on structures of size n with m relational tuples (denoted FO($\forall\exists\exists$)).

▶ **Theorem 3.** *For any $\epsilon > 0$, if there is an algorithm for detecting any HB race in time $O(\mathcal{N}^{1+\epsilon})$, then there is an algorithm for FO($\forall\exists\exists$) formulas in time $O(m^{1+\epsilon})$.*

It is known that FO($\forall\exists\exists$) can be solved in $O(m^{3/2})$ time [17], which yields a bound $O(n^3)$ for dense structures (i.e., when $m = \Theta(n^2)$). Theorem 3 implies that if $m^{3/2}$ is the best possible bound for FO($\forall\exists\exists$), then detecting any HB race cannot take $O(\mathcal{N}^{1+\epsilon})$ time for any $\epsilon < 1/2$. Although improvements for FO($\forall\exists\exists$) over the current $O(m^{3/2})$ bound might be possible, we find that a truly linear bound $O(m)$ would require major breakthroughs [1]. Under this hypothesis, Theorem 3 implies a super-linear bound for HB races.

Finally, we give an improved upper bound for this problem when $\mathcal{L} = o(\mathcal{T})$.

▶ **Theorem 4.** *Deciding whether σ has an HB race can be done in time $O(\mathcal{N} \cdot \min(\mathcal{T}, \mathcal{L}))$.*

In fact, similar to existing techniques [16], the algorithm behind Theorem 4 detects *all* variables that participate in an HB race (instead of just reporting σ as racy).

[1] Even the well-studied problem of testing triangle freeness, which is a special case of the similarly flavored FO($\exists\exists\exists$), has the super-linear bound $O(n^{\omega})$, where ω is the matrix multiplication exponent.

Synchronization-preserving races. Next, we turn our attention to the recently introduced sync-preserving races [25]. It is known that detecting sync-preserving races takes $O(\mathcal{N}\cdot\mathcal{V}\cdot\mathcal{T}^3)$ time. As sync-preserving races are known to be more expressive than HB races, the natural question is whether sync-preserving races can be detected more efficiently, e.g., by an algorithm that achieves a bound similar to Theorem 4 for HB races. Our next theorem answers this question in negative.

▶ **Theorem 5.** *For any $\epsilon > 0$, there is no algorithm that detects even a single sync-preserving race in time $O(\mathcal{N}^{3-\epsilon})$, unless the 3-OV hypothesis fails. Moreover, the statement holds even for traces over a single variable.*

As HB races take at most quadratic time, Theorem 5 shows that the increased expressiveness of sync-preserving races incurs a complexity overhead that is unavoidable in general.

Races based on the locking discipline. We now turn our attention to data races based on the locking discipline, namely *lock-cover races* and *lock-set races*. It is known that lock-cover races are more expressive than lock-set races. On the other hand, existing algorithms run in $O(\mathcal{N}^2\cdot\mathcal{L})$ time for lock-cover races and in $O(\mathcal{N}\cdot\mathcal{L})$ time for lock-set races, and thus hint that the former are computationally harder to detect. Our first theorem makes this separation formal, by showing that even with just two threads, having slightly more that logarithmically many locks implies a quadratic hardness for lock-cover races.

▶ **Theorem 6.** *For any $\epsilon > 0$, any $\mathcal{T} \geq 2$ and any $\mathcal{L} = \omega(\log \mathcal{N})$, there is no algorithm that detects even a single lock-cover race in time $O(\mathcal{N}^{2-\epsilon})$, unless the OV hypothesis fails.*

Observe that the $O(\mathcal{N}\cdot\mathcal{L})$ bound for lock-set races also becomes quadratic, when the number of locks is unbounded (i.e., $\mathcal{L} = \Theta(\mathcal{N})$). Is there a SETH-based quadratic lower bound similar to Theorem 6 for this case? Our next theorem rules out this possibility, again under NSETH.

▶ **Theorem 7.** *For any $\epsilon > 0$, there is no $(2^{\mathcal{N}}, \mathcal{N}^{1+\epsilon})$-fine-grained reduction from SAT to the problem of detecting any lock-set race, unless NSETH fails.*

Hence, even though we desire a quadratic lower bound, Theorem 7 rules out any super-linear lower-bound based on SETH. Alas, our next theorem shows that a quadratic lower bound for lock-set races does exist, based on the hardness of the hitting set (HS) problem.

▶ **Theorem 8.** *For any $\epsilon > 0$ and any $\mathcal{T} = \omega(\log n)$, there is no algorithm that detects even a single lock-set race in time $O(\mathcal{N}^{2-\epsilon})$, unless the HS hypothesis fails.*

Hitting set is a problem similar to OV, but has different quantifier structure. Just like the OV hypothesis, the HS hypothesis states that there is no sub-quadratic algorithm for the problem [3]. Although HS implies OV, the opposite is not known, and thus Theorem 8 does not contradict Theorem 7. In conclusion, we have that both lock-cover and lock-set races have (conditional) quadratic lower bounds, though the latter is based on a stronger hypothesis (HS), and requires more threads and locks for hardness to arise.

Finally, on our way to Theorem 7, we obtain the following theorem.

▶ **Theorem 9.** *Deciding whether a trace σ has a lock-set race on a given variable x can be performed in $O(\mathcal{N})$ time. Thus, deciding whether σ has a lock-set race can be performed in $O(\mathcal{N}\cdot\min(\mathcal{L},\mathcal{V}))$ time.*

Hence, Theorem 9 strengthens the $O(\mathcal{N}\cdot\mathcal{L})$ upper bound for lock-set races when $\mathcal{V} = o(\mathcal{L})$.

1.2 Related Work

Dynamic data-race detection. There exists a rich literature in dynamic techniques for data race detection. Methods based on vector clocks (Djit algorithm [19]) using Lamport's Happens Before (HB) [23] and the lock-set principle in Eraser [34] were the first ones to popularize dynamic analysis for detecting data races. Later work attempted to increase the performance of these notions using optimizations as in [30] and FastTrack [16], altogether different algorithms (e.g., the GoldiLocks algorithm [15]), and hybrid techniques [28]. HB and lock-set based race detection are respectively sound (but incomplete) and complete (but unsound) variants of the more general problem of data-race *prediction* [35]. While earlier work on data race prediction focused on explicit [35] or symbolic [32, 33] enumeration, recent efforts have focused on scalability [38, 24, 21, 29, 31, 39]. The more recent notion [26] of sync-preserving races generalizes the notion of HB. As the complexity of race prediction is prohibitive (NP-hard in general [25]), this work characterizes the fine-grained complexity of popular, more relaxed notions of dynamic races that take polynomial time.

Fine-grained complexity. Traditional complexity theory usually shows a problem is intractable by proving it NP-hard, and tractable by showing it is in P. For algorithms with large input sizes, this distinction may be too coarse. It becomes important to understand, even for problems in P, whether algorithms with smaller degree polynomials than the known are possible, or if there are fine-grained lower bounds making this unlikely. Fine-grained complexity involves proving such lower bounds, by showing relationships between problems in P, with an emphasis on the degree of the complexity polynomial, and is nowadays a field of very active study. We refer to [8] for an introductory, and to [43] for a more extensive exposition on the topic. Fine-grained arguments have also been instrumental in characterizing the complexity of various problems in concurrency, such as bounded context-switching [11], safety verification [12], data-race prediction [25] and consistency checking [13].

2 Preliminaries

2.1 Concurrent Program Executions and Data Races

Traces and Events. We consider execution traces (or simply *traces*) generated by concurrent programs, under the sequential consistency memory model. Under this memory model, a trace σ is a sequence of events. Each event e is labeled with a tuple $\mathsf{lab}(e) = \langle t, op \rangle$, where t is the (unique) identifier of the thread that performs the event e, and op is the operation performed in e. We will often write $e = \langle t, op \rangle$ instead of $\mathsf{lab}(e) = \langle t, op \rangle$. For the purpose of this presentation, an operation can be one of

(a) read ($\mathtt{r}(x)$) from, or write ($\mathtt{w}(x)$) to, a shared memory variable x, or
(b) acquire ($\mathtt{acq}(\ell)$) or release ($\mathtt{rel}(\ell)$) of a lock ℓ.

For an event $e = \langle t, op \rangle$, we use $\mathsf{tid}(e)$ and $\mathrm{op}(e)$ to denote respectively the thread identifier t and the operation op. For a trace σ, we use Events_σ to denote the set of events that appear in σ. Similarly, we will use $\mathsf{Threads}_\sigma$, Locks_σ and Vars_σ to denote respectively the set of threads, locks and shared variables that appear in trace σ. We denote by $\mathcal{N} = |\mathsf{Events}_\sigma|$, $\mathcal{T} = |\mathsf{Threads}_\sigma|$, $\mathcal{L} = |\mathsf{Locks}_\sigma|$, and $\mathcal{V} = |\mathsf{Vars}_\sigma|$. The set of read events and write events on variable $x \in \mathsf{Vars}_\sigma$ will be denoted by $\mathsf{Reads}_\sigma(x)$ and $\mathsf{Writes}_\sigma(x)$, and further we let $\mathsf{Accesses}_\sigma(x) = \mathsf{Reads}_\sigma(x) \cup \mathsf{Writes}_\sigma(x)$. Similarly, we let $\mathsf{Acquires}_\sigma(\ell)$ and $\mathsf{Releases}_\sigma(\ell)$ denote the set of lock-acquire and lock-release events, respectively, of σ on lock ℓ. The *trace order* of σ, denoted $\leq^\sigma_{\mathsf{tr}}$, is the total order on Events_σ induced by the sequence σ. Finally, the *thread-order* of σ, denoted $\leq^\sigma_{\mathsf{TO}}$ is the smallest partial order on Events_σ such that for any two events $e_1, e_2 \in \mathsf{Events}_\sigma$, if $e_1 \leq^\sigma_{\mathsf{tr}} e_2$ and $\mathsf{tid}(e_1) = \mathsf{tid}(e_2)$, then $e_1 \leq^\sigma_{\mathsf{TO}} e_2$.

Traces are assumed to be well-formed in that critical sections on the same lock do not overlap. For a lock $\ell \in \mathsf{Locks}_\sigma$, let $\sigma|_\ell$ be the projection of the trace σ on the set of events $\mathsf{Acquires}_\sigma(\ell) \cup \mathsf{Releases}_\sigma(\ell)$. Also, let $t_1, \dots t_k$ be the thread identifiers in $\mathsf{Threads}_\sigma$. Well-formedness then entails that for each lock ℓ, the projection $\sigma|_\ell$ is a prefix of some string in the language of the grammar with production rules $S \to \varepsilon | S \cdot S_{t_1} | S \cdot S_{t_2} | \cdots | S \cdot S_{t_k}$ and $S_{t_i} \to \langle t_i, \mathtt{acq}(\ell)\rangle \cdot \langle t_i, \mathtt{rel}(\ell)\rangle$ and start symbol S. Thus, every release event e has a unique matching acquire event, which we denote by $\mathsf{match}_\sigma(e)$. Likewise for an acquire event e, $\mathsf{match}_\sigma(e)$ denotes the unique matching release event if one exists. For an acquire event e, the critical section of e is the set of events $\mathsf{CS}_\sigma(e) = \{f \mid e \leq^\sigma_{\mathsf{TO}} f \leq^\sigma_{\mathsf{TO}} \mathsf{match}_\sigma(e)\}$ if $\mathsf{match}_\sigma(e)$ exists, and $\mathsf{CS}_\sigma(e) = \{f \mid e \leq^\sigma_{\mathsf{TO}} f\}$ otherwise.

Data Races. Two events $e_1, e_2 \in \mathsf{Events}_\sigma$ are said to be *conflicting* if they are performed by different threads, they are access events touching the same memory location, and at least one of them is a write access. Formally, we have (i) $\mathsf{tid}(e_1) \neq \mathsf{tid}(e_2)$, (ii) $e_1, e_2 \in \mathsf{Accesses}_\sigma(x)$ for some $x \in \mathsf{Vars}_\sigma$, and (iii) $\{e_1, e_2\} \cap \mathsf{Writes}_\sigma(x) \neq \varnothing$. An event $e \in \mathsf{Events}_\sigma$ is said to be *enabled* in a prefix ρ of σ, if for every event $e' \neq e$ with $e' \leq^\sigma_{\mathsf{TO}} e$, we have $e' \in \mathsf{Events}_\rho$. A data race in σ is a pair of conflicting events (e_1, e_2) such that there is a prefix ρ in which both e_1 and e_2 are simultaneously enabled.

2.2 Notions of Dynamic Data Races

As the problem of determining whether a concurrent program has an execution with a data race is undecidable, dynamic techniques observe program traces and report whether certain events indicate the presence of a race. Here we describe in detail some popular approaches to dynamic race detection that are the subject of this work.

Happens-Before Races. Given a trace σ, the *happens before* order $\leq^\sigma_{\mathsf{HB}}$ is the smallest partial order on Events_σ such that

(a) $\leq^\sigma_{\mathsf{TO}} \subseteq \leq^\sigma_{\mathsf{HB}}$, and

(b) for any lock $\ell \in \mathsf{Locks}_\sigma$ and for events $e \in \mathsf{Releases}_\sigma(\ell)$ and $f \in \mathsf{Acquires}_\sigma(\ell)$, if $e \leq^\sigma_{\mathsf{tr}} f$ then $e \leq^\sigma_{\mathsf{HB}} f$.

A pair of conflicting events (e_1, e_2) is an HB-race in σ if they are unordered by HB, i.e., $e_1 \not\leq^\sigma_{\mathsf{HB}} e_2$ and $e_2 \not\leq^\sigma_{\mathsf{HB}} e_1$. The associated decision question is, *given a trace σ, determine whether σ has an HB race.* Typically HB race detectors are tasked to report all events that form HB race with an earlier event in the trace [36, 2, 1]). That is, they solve the following function problem:*given a trace σ, determine all events $e_2 \in \mathsf{Events}_\sigma$ for which there exists an event $e_1 \in \mathsf{Events}_\sigma$ such that $e_1 \leq^\sigma_{tr} e_2$, and (e_1, e_2) is an HB race of σ.* The standard algorithm for solving both versions of the problem is a vector-clock algorithm that runs in $O(\mathcal{N} \cdot \mathcal{T})$ time [19].

Synchronization Preserving Races. Next, we present the notion of *sync(hronization)-preserving races* [25]. For a trace σ and a read event e, we use $\mathsf{lw}_\sigma(e)$ to denote the write event observed by e. That is, $e' = \mathsf{lw}_\sigma(e)$ is the last (according to the trace order $\leq^\sigma_{\mathsf{tr}}$) write event e' of σ such that e and e' access the same variable and $e' \leq^\sigma_{\mathsf{tr}} e$; if no such e' exists, then we write $\mathsf{lw}_\sigma(e) = \bot$. A trace ρ is said to be a correct reordering of trace σ, if

(a) $\mathsf{Events}_\rho \subseteq \mathsf{Events}_\sigma$

(b) Events_ρ is downward closed with respect to $\leq^\sigma_{\mathsf{TO}}$, and further $\leq^\rho_{\mathsf{TO}} \subseteq \leq^\sigma_{\mathsf{TO}}$, and

(c) for every read event $e \in \mathsf{Events}_\rho$, $\mathsf{lw}_\rho(e) = \mathsf{lw}_\sigma(e)$.

	t_1	t_2
1	acq(ℓ)	
2	w(x)	
3	rel(ℓ)	
4		w(x)
5		acq(ℓ)
6		rel(ℓ)

(a) HB-race.

	t_1	t_2
1	w(x)	
2	acq(ℓ)	
3	rel(ℓ)	
4		acq(ℓ)
5		rel(ℓ)
6		w(x)

(b) Sync-preserving race.

	t_1	t_2
1	acq(ℓ)	
2	w(x)	
3	rel(ℓ)	
4		acq(ℓ)
5		r(x)
6		rel(ℓ)
7		w(x)

(c) Lock-cover race.

	t_1	t_2
1	acq(ℓ_1)	
2	acq(ℓ_2)	
3	w(x)	
4	rel(ℓ_2)	
5	rel(ℓ_1)	
6		acq(ℓ_2)
7		acq(ℓ_3)
8		w(x)
9		rel(ℓ_3)
10		rel(ℓ_2)
11	acq(ℓ_1)	
12	acq(ℓ_3)	
13	w(x)	
14	rel(ℓ_3)	
15	rel(ℓ_1)	

(d) Lockset race.

Figure 1 Types of data races.

Further, ρ is *sync-preserving* with respect to σ if for every lock ℓ and for any two acquire events $e_1, e_2 \in \mathsf{Acquires}_\rho(\ell)$, we have $e_1 \leq^\rho_{\mathsf{tr}} e_2$ iff $e_1 \leq^\sigma_{\mathsf{tr}} e_2$. Thus, the order of critical sections on the same lock is the same in σ and ρ.

A pair of conflicting events (e_1, e_2) is a *sync-preserving race* in σ if σ has a sync-preserving correct reordering ρ such that (e_1, e_2) is a data race of ρ. The associated decision question is, *given a trace σ, determine whether σ has a sync-preserving race.* As with HB races, we are typically interested in reporting all events $e_2 \in \mathsf{Events}_\sigma$ for which there exists an event $e_1 \in \mathsf{Events}_\sigma$ such that $e_1 \leq^\sigma_{\mathsf{tr}} e_2$, and (e_1, e_2) is an sync-preserving race of σ. It is known one can report all such events e_2 in time $O(\mathcal{N} \cdot \mathcal{V} \cdot \mathcal{T}^3)$.

Lock-Cover and Lock-Set Races. Lock-cover and lock-set races indicate violations of the *locking discipline*. For an event e in a trace σ, let $\mathsf{locksHeld}_\sigma(e) = \{\ell \,|\, \exists f \in \mathsf{Acquires}_\sigma(\ell),$ such that $e \in \mathsf{CS}_\sigma(f)\}$, i.e., $\mathsf{locksHeld}_\sigma(e)$ is the set of locks held by thread $\mathsf{tid}(e)$ when e is executed. A pair (e_1, e_2) of conflicting events might indicate a data race if $\mathsf{locksHeld}_\sigma(e_1) \cap \mathsf{locksHeld}_\sigma(e_2) = \varnothing$. Although this condition doesn't guarantee the presence of a race, it constitutes a violation of the locking discipline and can be further investigated.

A pair of conflicting events (e_1, e_2) is a *lock-cover race* if $\mathsf{locksHeld}_\sigma(e_1) \cap \mathsf{locksHeld}_\sigma(e_2) = \varnothing$. The decision question is, *given a trace σ, determine if σ has a lock-cover race.* The problem is solvable in $O(\mathcal{N}^2 \cdot \mathcal{L})$ time, by checking the above condition over all conflicting event pairs.

As the algorithm for lock-cover races takes quadratic time, developers often look for less expensive indications of violations of locking discipline, called lock-set races (as proposed by ERASER race detector [34]). A trace σ has a *lock-set race* on variable $x \in \mathsf{Vars}_\sigma$ if

(a) there exists a pair of conflicting events $(e_1, e_2) \in \mathsf{Writes}_\sigma(x) \times \mathsf{Accesses}_\sigma(x)$, and

(b) $\bigcap_{e \in \mathsf{Accesses}_\sigma(x)} \mathsf{locksHeld}_\sigma(e) = \varnothing$.

The associated decision question is, *given a trace σ, determine if σ has a lock-set race.* Note that a lock-cover race implies a lock-set race, but not vice versa. On the other hand, determining whether σ has a lock-set race is easily performed in $O(\mathcal{N} \cdot \mathcal{L})$ time.

Example. We illustrate the different notions of races in Figure 1. We use e_i to denote the i^{th} event of the trace in consideration. First consider the trace σ_a in Figure 1a. The events e_2 and e_4 are conflicting and unordered by $\leq^{\sigma_a}_{\mathsf{HB}}$, thus (e_2, e_4) is an HB-race. Second, in trace σ_b

of Figure 1b, the pair (e_1, e_6) is not an HB-race as $e_1 \leq_{\mathsf{HB}}^{\sigma_b} e_6$. But this is a sync-preserving race witnessed by the correct reordering e_4, e_5, as both e_1 and e_6 are enabled. Third, in trace σ_c of Figure 1c, the pair (e_2, e_7) is neither a sync-preserving race nor an HB race, but is a lock-cover race as $\mathsf{locksHeld}_{\sigma_c}(e_2) \cap \mathsf{locksHeld}_{\sigma_c}(e_7) = \varnothing$. Finally, the trace σ_d in Figure 1d has no HB, sync-preserving or lock-cover race, as all $\mathtt{w}(x)$ are protected by a common lock. But there is a lock-set race on x as there is no single lock that protects all $\mathtt{w}(x)$.

2.3 Fine-Grained Complexity and Popular Hypotheses

In this section we present notions of fine-grained complexity theory that are relevant to our work. We refer to the survey [43] for a detailed exposition on the topic. This theory relates the computational complexity of problems under the popular notion of fine-grained reductions (See Appendix A for a formal definition).

Such a reduction $\mathrm{A}_{(\mathsf{a})}\preceq_{(\mathsf{b})}\mathrm{B}$ would be interesting for B if $a(n)$ was a proven or well-believed conjectured lower bound on A, thus implying a believable lower bound on B. One such well-believed conjecture in complexity theory is SETH [18] (See Appendix A for a formal definition) for the classic CNF-SAT problem.SETH implies a lower bound conjecture, denoted by OVH, on the Orthogonal Vectors problem OV, as shown by a reduction from CNF-SAT to k-OV [42]. Thus, a conditional lower bound under OVH implies one under SETH as well, leading to numerous conditional lower bound results under OVH [See [43] for a detailed literature review]. We next formally define k-OV and OVH.

An instance of k-OV is an integer $d = \omega(\log n)$ and k sets $A_i \subseteq \{0,1\}^d,\ i \in [k]$ such that $|A_i| = n$, and denoted by $\mathsf{OV}(n, d, k)$.

▶ **Problem 1** (Orthogonal Vectors (k-OV)). *Given an instance OV(n, d, k), the k-OV problem is to decide if there are k vectors $a_i \in A_i$ for all $i \in [k]$ such that the sum of their point wise product is zero, i.e., $\sum_{j=1}^{d} \prod_{i=1}^{k} a_i[j] = 0$.*

For ease of exposition, we denote $\mathsf{OV}(n, d, 2)$ and 2-OV by $\mathsf{OV}(n, d)$ and OV respectively.

▶ **Hypothesis 1** (Orthogonal Vectors Hypothesis (OVH)). *No randomized algorithm can solve k-OV for an instance OV(n, d, k) in time $O(n^{(k-\epsilon)} \cdot \mathrm{poly}(d))$ for any constant $\epsilon > 0$.*

The following impossibility result from [9] proves that a reduction under SETH, and hence under OVH, is not possible unless the NSETH conjecture (definition included in Appendix A) is false.

▶ **Theorem 10.** *If NSETH holds and a problem $C \in$ NTIME$[\mathsf{T_C}] \cap$ coNTIME$[\mathsf{T_C}]$, then for any problem B that is SETH-hard under deterministic reductions with time T_B, and $\gamma > 0$, we cannot have a fine-grained reduction $B\ _{(\mathsf{T_B})}\preceq_{(\mathsf{c})}\ C$ where $c = T_C^{(1+\gamma)}$.*

We show some of our problems satisfy the conditions of Theorem 10, and hence show lower bounds for these conditioned on two other hypotheses described below.

An instance of the hitting set problem, denoted by HS, is an integer $d = \omega(\log n)$ and sets $X, Y \subseteq \{0,1\}^d,\ i \in [n]$ such that $|X| = |Y| = n$, and denoted by HS(n,d).

▶ **Problem 2** (Hitting Sets (HS)). *Given an instance HS(n,d), the HS problem is to decide if there is a vector $x \in X$ such that for all $y \in Y$ we have $x \cdot y \neq 0$, or informally, some vector in X hits all vectors in Y.*

▶ **Hypothesis 2** (Hitting Sets Hypothesis (HSH)). *No randomized algorithm can solve HS for an instance HS(n,d) in time $O(n^{(2-\epsilon)} \cdot \mathrm{poly}(d))$ for any constant $\epsilon > 0$.*

HSH implies OVH, but the reverse direction is not known.

Finally we consider the subclass of first order formulae over structures of size n and with m relational tuples [17].

▶ **Problem 3** (FO(∀∃∃))**.** *Decide if a given a first-order formula quantified by ∀∃∃ has a model on a structure of size n with m relational tuples.*

It is known that FO(∀∃∃) can be solved in $O(m^{3/2})$ time using ideas from triangle detection algorithms [17]. For dense graph structures ($m = \Theta(n^2)$), this yields the bound $O(n^3)$. Although sub-cubic algorithms might be possible, achieving a truly quadratic bound seems unlikely or at least highly non-trivial.

3 Happens-Before Races

In this section we prove the results for detecting HB races, i.e., Theorem 1 to Theorem 4.

3.1 Algorithm for HB Races

We first outline our $O(\mathcal{N} \cdot \mathcal{L})$-time algorithm for checking if a trace σ has an HB-race, thereby proving Theorem 4. As with the standard vector clock algorithm [19], our algorithm is based on computing timestamps for each event. However, unlike the standard algorithm that assigns thread-indexed timestamps, we use *lock-indexed* timestamps, or *lockstamps*, which we formalize next. We fix the input trace σ in the rest of the discussion.

Lockstamps. A lockstamp is a mapping from locks to natural numbers (including infinity) $L : \mathsf{Locks}_\sigma \to \mathbb{N} \cup \{\infty\}$. Given lockstamps L, L_1, L_2 and lock ℓ, we use the notation

(i) $L[\ell \mapsto c]$ to denote the the lockstamp $\lambda m \cdot$ if $m = \ell$ then c else $L(m)$,

(ii) $L_1 \sqcup L_2$ to denote the pointwise maximum, i.e., $(L_1 \sqcup L_2)(\ell) = \max(L_1(\ell), L_2(\ell))$ for every ℓ,

(iii) $L_1 \sqcap L_2$ to denote the pointwise minimum, and

(iv) $L_1 \sqsubseteq L_2$ to denote the predicate $\forall \ell \cdot L_1(\ell) \leq L_2(\ell)$.

Our algorithm computes *acquire* and *release* lockstamps AcqLS^σ_e and RelLS^σ_e for every event $e \in \mathsf{Events}_\sigma$, defined next. For a lock ℓ and acquire event $f \in \mathsf{Acquires}_\sigma(\ell)$ (resp. release event $g \in \mathsf{Releases}_\sigma(\ell)$), let $pos_\sigma(f) = |\{f' \in \mathsf{Acquires}_\sigma(\ell) \,|\, f' \leq^\sigma_{\mathsf{tr}} f\}|$ (resp. $pos_\sigma(g) = |\{g' \in \mathsf{Releases}_\sigma(\ell) \,|\, g' \leq^\sigma_{\mathsf{tr}} g\}|$) denote the relative position of f (resp. g) among all acquire events (resp. release events) of ℓ. Then, for an event $e \in \mathsf{Events}_\sigma$ the lockstamps AcqLS^σ_e and RelLS^σ_e are defined as follows (we assume that $\max \varnothing = 0$ and $\min \varnothing = \infty$.)

$$\begin{aligned} \mathsf{AcqLS}^\sigma_e(\ell) &= \lambda \ell \cdot \max\{pos_\sigma(f) \,|\, f \in \mathsf{Acquires}_\sigma(\ell), f \leq^\sigma_{\mathsf{HB}} e\} \\ \mathsf{RelLS}^\sigma_e(\ell) &= \lambda \ell \cdot \min\{pos_\sigma(g) \,|\, g \in \mathsf{Releases}_\sigma(\ell), e \leq^\sigma_{\mathsf{HB}} g\} \end{aligned} \tag{1}$$

Our $O(\mathcal{N} \cdot \mathcal{L})$ algorithm now relies on the following observations. First, the HB partial order can be inferred by comparing lockstamps of events (Lemma 11). Second, there is an $O(\mathcal{N} \cdot \mathcal{L})$ time algorithm that computes the acquire and release lockstamps for each event in the input trace. Third, the existence of an HB race can be determined by examining only $O(\mathcal{N})$ pairs of conflicting events (using their lockstamps), instead of all possible $O(\mathcal{N}^2)$ pairs (Lemma 12). Finally, we can also examine all the $O(\mathcal{N})$ pairs in time $O(\mathcal{N} \cdot \mathcal{L})$ (using $O(\mathcal{N})$ lockstamp comparisons) and thus determine the existence of an HB race in the same asymptotic running time. Let us first state how we use lockstamps to infer the HB relation.

▶ **Lemma 11.** *Let $e_1 \leq^\sigma_{tr} e_2$ be events in σ such that* $\mathsf{tid}(e_1) \neq \mathsf{tid}(e_2)$*. We have, $e_1 \leq^\sigma_{HB} e_2 \iff \neg(\mathsf{AcqLS}^\sigma_{e_2} \sqsubseteq \mathsf{RelLS}^\sigma_{e_1})$*

The proof of Lemma 11 is presented in Appendix B.1.

Computing Lockstamps. We now illustrate how to compute the acquire lockstamps for all events, by processing the trace σ in a forward pass. For each thread t and lock ℓ, we maintain lockstamp variables $\mathbb{C}_t$ and $\mathbb{L}_\ell$. We also maintain an integer variable p_ℓ for each lock ℓ that stores the index of the latest $\mathtt{acq}(\ell)$ event in σ. Initially, we set $\mathbb{C}_t$ and $\mathbb{L}_m$ to the *bottom* map $\lambda\ell \cdot 0$, and p_m to 0, for each thread t and lock m. We traverse σ left to right, and perform updates to the data structures as described in Algorithm 1, by invoking the appropriate *handler* based on the thread and operation of the current event $e = \langle t, op\rangle$. At the end of each handler, we assign the lockstamp AcqLS^σ_e to e. The computation of release lockstamps is similar, albeit in a reverse pass, and presented in Appendix B.1. Observe that each step takes $O(\mathcal{L})$ time giving us a total running time of $O(\mathcal{N} \cdot \mathcal{L})$ to assign lockstamps.

Algorithm 1 *Assigning acquire lockstamps to events in the trace.*

```
acquire(t, ℓ):
    $\mathsf{p}_\ell \leftarrow \mathsf{p}_\ell + 1$
    $\mathbb{C}_t \leftarrow \mathbb{C}_t[\ell \mapsto \mathsf{p}_\ell] \sqcup \mathbb{L}_\ell$
    $\mathsf{AcqLS}^\sigma_e \leftarrow \mathbb{C}_t$

release(t, ℓ):
    $\mathbb{L}_\ell \leftarrow \mathbb{C}_t$
    $\mathsf{AcqLS}^\sigma_e \leftarrow \mathbb{C}_t$

read(t, x):
    $\mathsf{AcqLS}^\sigma_e \leftarrow \mathbb{C}_t$

write(t, x):
    $\mathsf{AcqLS}^\sigma_e \leftarrow \mathbb{C}_t$
```

We say that a pair of conflicting access events (e_1, e_2) (with $e_1 \leq^\sigma_{\mathsf{tr}} e_2$) to a variable x is a *consecutive conflicting pair* if there is no event $f \in \mathsf{Writes}_\sigma(x)$ such that $e_1 <^\sigma_{\mathsf{tr}} f <^\sigma_{\mathsf{tr}} e_2$. We make the following observation.

▶ **Lemma 12.** *A trace σ has an HB-race iff there is pair of consecutive conflicting events in σ that is an HB-race. Moreover, σ has at most $O(\mathcal{N})$ many consecutive conflicting pairs of events.*

Checking for an HB race. We now describe the algorithm for checking for an HB race in σ. We perform a forward pass on σ while storing the release lockstamps of some of the earlier events. When processing an access event e, we check if it is in race with an earlier event by comparing the acquire lockstamp of e with a previously stored release lockstamp. More precisely, we maintain a variable $\mathbb{W}_x$ to store the release lockstamp of the last write event on x, a variable t^w_x to store the thread that performed this write and set S_x to store pairs (t, L) of threads and release lockstamps of all the read events performed since the last write on x was observed. Initially, $\mathsf{t}^w_x = \mathtt{NIL}$, $\mathbb{W}_x = \lambda\ell \cdot \infty$ and $\mathsf{S}_x = \varnothing$. The updates performed at each read or write event e are presented in the corresponding handler in Algorithm 2; no updates need to be performed at acquire or release events in this case.

Algorithm 2 *Determining the existence of an HB-race using lockstamps.*

```
read(t, x):
    if $\mathsf{t}^w_x \notin \{NIL, t\} \wedge \mathsf{AcqLS}^\sigma_e \sqsubseteq \mathbb{W}_x$ then
        declare "race" and exit
    $\mathsf{S}_x \leftarrow \mathsf{S}_x \cup \{(t, \mathsf{RelLS}^\sigma_e)\}$

write(t, x):
    if $\mathsf{t}^w_x \notin \{NIL, t\} \wedge \mathsf{AcqLS}^\sigma_e \sqsubseteq \mathbb{W}_x$ then
        declare "race" and exit
    if $\exists (u, L) \in \mathsf{S}_x, t \neq u \wedge \mathsf{AcqLS}^\sigma_e \sqsubseteq L$ then
        declare "race" and exit
    $\mathsf{t}^w_x = t$; $\mathsf{S}_x \leftarrow \varnothing$; $\mathbb{W}_x \leftarrow \mathsf{RelLS}^\sigma_e$
```

We refer to Appendix B.1 for the correctness, which concludes the proof of Theorem 4.

3.2 Hardness Results for HB

We now turn our attention to the hardness results for HB race detection. To this end, we prove Theorem 1, Theorem 2, and Theorem 3. We start with defining the graph $\mathsf{G}^{\sigma}_{\mathsf{HB}}$, which can be thought of as a form of transitive reduction of the HB relation.For an integer $n \geq 1$, we use $[n]$ to denote $\{1, \dots, n\}$.

The graph $\mathsf{G}^{\sigma}_{\mathsf{HB}}$. Given a trace σ, the graph $\mathsf{G}^{\sigma}_{\mathsf{HB}}$ is a directed graph with node set Events_{σ}, and we have an edge (e_1, e_2) in $\mathsf{G}^{\sigma}_{\mathsf{HB}}$ iff

(i) e_2 is the immediate successor of e_1 in thread order $\leq^{\sigma}_{\mathsf{TO}}$, or

(ii) $e_1 \in \mathsf{Acquires}_{\sigma}(\ell)$, $e_2 \in \mathsf{Releases}_{\sigma}(\ell)$, $e_1 \leq^{\sigma}_{\mathsf{tr}} e_2$, and there is no intermediate event in σ that accesses the common lock ℓ.

It follows easily that for any two distinct events e_1, e_2, we have $e_1 \leq^{\sigma}_{\mathsf{HB}} e_2$ iff e_2 is reachable from e_1 in $\mathsf{G}^{\sigma}_{\mathsf{HB}}$. Moreover, every node has out-degree ≤ 2 and thus $\mathsf{G}^{\sigma}_{\mathsf{HB}}$ is *sparse*, while it can be easily constructed in $O(\mathcal{N})$ time.

OV hardness of write-read HB races

Given a OV instance $\mathsf{OV}(n, d)$ on two vector sets A_1, A_2, we create a trace σ as follows. For the part A_1 of OV, we introduce $n{\cdot}(d+1)$ threads $\{t_{(x,i)}\}_{x \in A_1, i \in \{0\} \cup [d]}$, and d locks $\{l_i\}_{i \in [d]}$. For the second part A_2 we introduce $n{\cdot}d$ locks denoted by $\{l_{(y,i)}\}_{y \in A_2, i \in [d]}$, and n threads $\{t_y\}_{y \in A_2}$. Finally, we have a single variable z.

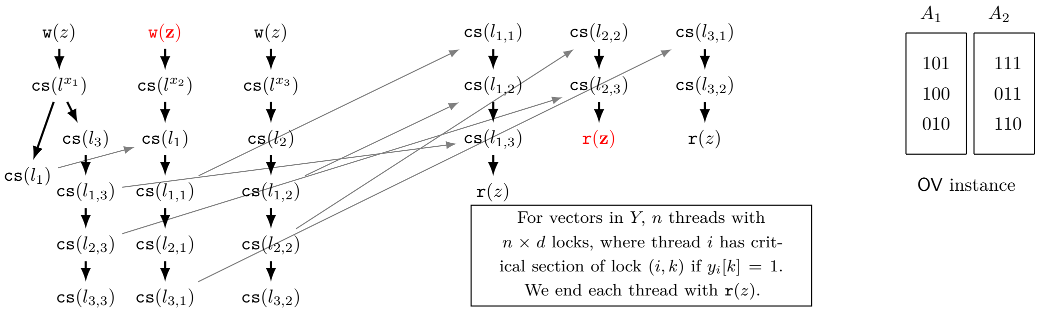
We first describe the threads $t(x, i)$. For each vector x, for each $i \in [d]$ with $x[i] = 1$, we introduce a critical section on the lock l_i. If x is the last vector of A_1 with $x[i] = 1$, we also insert the critical sections $l_{(y,i)}$ for all $y \in [n]$, to $t(x, i)$ after the critical section of l_x. Finally, we construct a thread $t_{x,0}$ which starts with a write event $\mathtt{w}(z)$, followed by a critical section on lock l^x. We also insert a critical section on lock l^x to all threads $t(x, i)$, for $i \in [d]$. Hence the $\mathtt{w}(z)$ event is ordered by HB before all other events of $t(x, i)$.

Now we describe the threads t_y. For each $i \in [d]$, if $y[i] = 1$, we add a critical section of the lock $l(y, i)$ in t_y. We end the thread with a read event $\mathtt{r}(z)$.

Finally, we construct σ by first executing each thread $t(x, i)$ in the pre-determined order of $x \in A_1$, followed by executing the traces t_y in any order. See Figure 2 for an illustration. We refer to Appendix B for the correctness, which concludes the proof of Theorem 1.

Conditional impossibility for SETH-based hardness

We now turn our attention to the problem of detecting a single HB race (i.e., not necessarily involving a read event). We define a useful multi-connectivity problem on graphs.



For vectors in Y, n threads with $n \times d$ locks, where thread i has critical section of lock (i, k) if $y_i[k] = 1$. We end each thread with $\mathtt{r}(z)$.

Figure 2 Reducing OV to finding HB races. For simplicity, we show the graph $\mathsf{G}^{\sigma}_{\mathsf{HB}}$ instead of the trace σ. The HB race is marked in red, corresponding to the orthogonal pair (x_2, y_2).

▶ **Problem 4** (MCONN). *Given a directed graph G with n nodes and m edges, and k pairs of nodes $(s_i, t_i), i \in [k]$, decide if there is a path in G from every s_i to the corresponding t_i.*

Due to Lemma 12, detecting whether there is an HB race in σ reduces to testing MCONN between all $O(\mathcal{N})$ pairs of consecutive conflicting events in σ.

Short witnesses for HB races. We now prove Theorem 2. Following [9, Corollary 2], it suffices to show that deciding MCONN can be done in $\mathsf{NTIME}[\mathcal{N}^{3/2}] \cap \mathsf{coNTIME}[\mathcal{N}^{3/2}]$. At a first glance, the bound $\mathsf{NTIME}[\mathcal{N}^{3/2}]$ may seem too optimistic, as there are $\Theta(\mathcal{N})$ paths $P_i \colon s_i \rightsquigarrow t_i$, and each of them can have size $\Theta(\mathcal{N})$. Hence even just guessing these paths appears to take quadratic time. Our proof shows that more succinct witnesses exist.

Proof of Theorem 2. First consider the simpler case where σ has an HB-race. Phrased as a MCONN problem on $\mathsf{G}^{\sigma}_{\mathsf{HB}}$, it suffices to show that there is a pair (s_i, t_i) such that s_i does not reach t_i. We construct a non-deterministic algorithm for this task that simply guesses the pair (s_i, t_i), and verifies that there is no $s_i \rightsquigarrow t_i$ path. Since $\mathsf{G}^{\sigma}_{\mathsf{HB}}$ is sparse, this can be easily verified in $O(\mathcal{N})$ time.

Now consider the case when there is no HB-race. Phrased as a MCONN problem on $\mathsf{G}^{\sigma}_{\mathsf{HB}}$, it suffices to verify that for every pair (s_i, t_i), we have that s_i reaches t_i. We construct a non-deterministic algorithm for this task, as follows. The algorithm operates in two phases, using a set A, initialized as $A = \{(s_i, t_i)\}_{i \in k}$.

1. In the first phase, the algorithm repeatedly guesses a node u that lies on at least $\mathcal{N}^{1/2}$ paths $s_i \rightsquigarrow t_i$, for $(s_i, t_i) \in A$. It verifies this guess via a backward and a forward traversal from u. The algorithm then removes all such (s_i, t_i) from A, and repeats.
2. In the second phase, the algorithm guesses for every remaining $(s_i, t_i) \in A$ a path $P_i \colon s_i \rightsquigarrow t_i$, and verifies that P_i is a valid path.

Phase 1 can be execute at most $\mathcal{N}^{1/2}$ iterations, while each iteration takes $O(\mathcal{N})$ time since $\mathsf{G}^{\sigma}_{\mathsf{HB}}$ is sparse. Hence the total time for phase 1 is $O(\mathcal{N}^{3/2})$. Phase 2 takes $O(\mathcal{N}^{3/2})$ time, as every node of $\mathsf{G}^{\sigma}_{\mathsf{HB}}$ appears in at most $\mathcal{N}^{1/2}$ paths P_i. The desired result follows. ◀

A super-linear lower bound for general HB races

Finally, we turn our attention to Theorem 3. The problem $\mathsf{FO}(\forall\exists\exists)$ takes as input a first-order formula ϕ with quantifier structure $\forall\exists\exists$ and whose atoms are tuples, and the task is to verify whether ϕ has a model on a structure of n elements and m relational tuples. For simplicity, we can think of the structure as a graph G of n nodes and m edges, and ϕ a formula that characterizes the presence/absence of edges (e.g., $\phi = \forall x \exists y \exists z \; e(x, y) \wedge \neg e(y, z)$).

The crux of the proof of Theorem 3 is showing the following lemma.

▶ **Lemma 13.** *$\mathsf{FO}(\forall\exists\exists)$ reduces to MCONN on a graph G with $O(n)$ nodes in $O(n^2)$ time.*

Finally, we arrive at Theorem 3 by constructing in $O(n^2)$ time a trace σ with $\mathcal{N} = \Theta(n^2)$ such that $\mathsf{G}^{\sigma}_{\mathsf{HB}}$ is similar in structure to the graph G of Lemma 13. In the end, detecting an HB race in σ in $O(\mathcal{N}^{1+\epsilon})$ time yields an algorithm for $\mathsf{FO}(\forall\exists\exists)$ in $\Theta(n^{2+\epsilon'})$ time. We refer to Appendix B for the details, which conclude the proof of Theorem 3.

4 Synchronization-Preserving Races

In this section, we discuss the dynamic detection of sync-preserving races, and prove Theorem 5.

For notational convenience, we will frequently use the composite *sync* events. A $\mathtt{sync}(\ell)$ event represents the sequence $\mathtt{acq}(\ell), \mathtt{r}(x_\ell), \mathtt{w}(x_\ell), \mathtt{rel}(\ell)$. The key ideas behind the sync events are as follows. First, if X_ℓ appears only in $\mathtt{sync}(\ell)$ events, then there can be no race involving these. Second, assume that in a trace σ we have two $\mathtt{sync}(\ell)$ events e_1 and e_2 with $e_1 <^{\sigma}_{\mathsf{tr}} e_2$. Then any correct reordering ρ of σ with $e_2 \in \mathsf{Events}_\rho$ satisfies the following.

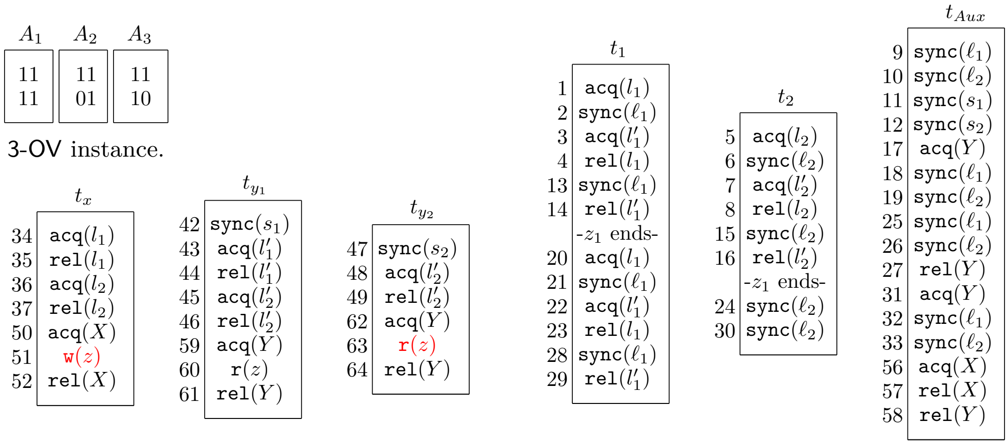


Figure 3 Example reduction from 3-OV to sync race detection. The trace orders events as shown by their numbering. We only show one thread t_x, as the two x vectors are identical.

(a) We have $e_1 \in \mathsf{Events}_\rho$. This is because, for any two consecutive $\mathtt{sync}(\ell)$ events $e \leq^{\sigma}_{\mathsf{tr}} e'$, $\mathsf{lw}_\sigma(e'_{\mathtt{r}}) = e_{\mathtt{w}}$, where $e'_{\mathtt{r}}$ is the $\mathtt{r}(x_\ell)$ event in the sync sequence e, and $e_{\mathtt{w}}$ is the $\mathtt{w}(x_\ell)$ event in the sync sequence e.

(b) For every $e'_1, e'_2 \in \mathsf{Events}_\rho$ such that $e'_1 \leq^{\sigma}_{\mathsf{TO}} e_1$ and $e_2 \leq^{\sigma}_{\mathsf{TO}} e'_2$, we have $e'_1 <^{\rho}_{\mathsf{tr}} e'_2$.

We hence use sync events to ensure certain orderings in any sync-preserving correct reordering of σ that exposes a sync-preserving data race.

Informal Description

Before we proceed with the detailed reduction, we provide a high-level description. The input to 3-OV is three sets of vectors $A_1 = \{x_i\}_{i\in[n]}$, $A_2 = \{y_i\}_{i\in[n]}$, and $A_3 = \{z_i\}_{i\in[n]}$. Every vector $x \in A_1$ is represented by a thread t_x, ending with the critical section $\mathtt{acq}(X), \mathtt{w}(z), \mathtt{rel}(X)$. Similarly, every vector $y \in A_2$ is represented by a thread t_y, ending with the critical section $\mathtt{acq}(Y), \mathtt{r}(z), \mathtt{rel}(Y)$. There are no further access events, hence we can only have a race between the write event of a thread t_x and the read event of a thread t_y. To encode the vectors in A_3, we use k threads t_k, for $k \in [d]$. Each thread t_k has n *segments* such that the i^{th} segment of t_k encodes $z_i[k]$. Finally, there is a single thread t that will have the property (enforced using sync events) that it must be included in any reordering if and only if all the threads encoding A_3 are included entirely. The thread t also has locks that, if present in a reordering, prevent the access events of z from being in race with each other.

If there is a triplet of vectors $x \in A_1$, $y \in A_2$ and $z \in A_3$ that is orthogonal, then a valid reordering of the trace σ need only contain the threads corresponding to A_3 up to the events of z; the thread t is not required to be a part of this reordering, causing the events of x and y to be in race. If no such triple exists, then the notion of sync-preservation ensures that all events of the threads representing A_3 must be present in any valid reordering of σ, thus enforcing t also to be a part of such a reordering. Thus, the access events belonging to some threads t_x and t_y will be in race if and only if there is a vector $z \in A_3$ that makes the triplet x, y, z orthogonal.

Reduction

Given a 3-OV instance OV(n, d, 3) on vector sets $A_1 = \{x_i\}_{i\in[n]}$, $A_2 = \{y_i\}_{i\in[n]}$, and $A_3 = \{z_i\}_{i\in[n]}$, we create a trace σ as follows (see Figure 3). We have $\mathcal{T} = 2 \cdot n + d + 1$ threads, while all access events (not counting the sync events) are of the form $\mathtt{w}(z)/\mathtt{r}(z)$ in a single variable z. We first describe the threads, and then how they interleave in σ.

Threads. We introduce a thread t_x for every vector $x \in A_1$ and a lock l_k for every $k \in [d]$. Each thread t_x consists of two segments t_x^1 and t_x^2. We create t_x^1 as follows. For every $k \in [d]$ where $x[k] = 1$, we add an empty critical section $\texttt{acq}(l_k), \texttt{rel}(l_k)$ in t_x^1. We create t_x^2 as the sequence $\texttt{acq}(X), \texttt{w}(z), \texttt{rel}(X)$, where X is a new lock, common for all t_x^2.

For the vectors in A_2, we introduce threads similar to those of part A_1, as follows. We have a thread t_y for every vector $y \in A_2$ and a lock l'_k for every $k \in [d]$. Each thread t_y consists of two segments t_y^1 and t_y^2. For every $k \in [d]$ where $y[k] = 1$, we add an empty critical section $\texttt{acq}(l'_k), \texttt{rel}(l'_k)$ in t_y^1. In contrast to the t_x^1, every t_y^1 also has an event $\texttt{sync}(s_y)$ at the very beginning. We create t_y^2 as the sequence $\texttt{acq}(Y), \texttt{r}(z), \texttt{rel}(Y)$, where Y is a new lock, common for all t_y^2.

The construction of the threads corresponding to the vectors in A_3 is more involved. We have one thread t_k for every $k \in [d]$. Each thread has some fixed $\texttt{sync}$ events, as well as critical sections corresponding to one coordinate of all n vectors in A_3. In particular, we construct each t_k as follows. We iterate over all z_i, and if $z_i[k] = 0$, we simply append two events $\texttt{sync}(\ell_k), \texttt{sync}(\ell_k)$ to t_k. On the other hand, if $z_i[k] = 1$, we interleave these sync events with two critical sections, by appending the sequence $\texttt{acq}(l_k), \texttt{sync}(\ell_k), \texttt{acq}(l'_k), \texttt{rel}(l_k), \texttt{sync}(\ell_k), \texttt{rel}(l'_k)$.

Lastly, we have a single auxiliary thread t that consists of three parts t^1, t^2 and t^3, where

$$t^1 = \texttt{sync}(\ell_1), \dots, \texttt{sync}(\ell_k), \texttt{sync}(s_{y_1}), \dots \texttt{sync}(s_{y_n})$$
$$t^2 = (\texttt{acq}(Y), \texttt{sync}(\ell_1), \dots, \texttt{sync}(\ell_k), \texttt{sync}(\ell_1), \dots, \texttt{sync}(\ell_k), \texttt{rel}(Y))^{n-1}$$
$$t^3 = \texttt{acq}(Y), \texttt{sync}(\ell_1), \dots, \texttt{sync}(\ell_k), \texttt{acq}(X), \texttt{rel}(X), \texttt{rel}(Y)$$

Concurrent trace. We are now ready to describe the interleaving of the above threads in order to obtain the concurrent trace σ.

1. We execute the auxiliary thread t and all threads t_k, for $k \in [d]$ (i.e., the threads corresponding to the vectors of A_3) arbitrarily, as long as for every $k \in [d]$, every sequence of $\texttt{sync}(\ell_k)$ events
   (a) starts with the $\texttt{sync}(\ell_k)$ event of t_k and proceeds with the $\texttt{sync}(\ell_k)$ event of t,
   (b) strictly alternates in every two $\texttt{sync}(\ell_k)$ events between t and t_k, and
   (c) ends with the last $\texttt{sync}(\ell_k)$ event of t_k.
2. We execute all t_x^1 and t_y^1 (i.e., the first parts of all threads that correspond to the vectors in A_1 and A_2) arbitrarily, but after all threads t_k, for $k \in [d]$.
3. We execute all t_x^2 (i.e., the second parts of all threads that correspond to the vectors in A_1) arbitrarily, but before the segment $\texttt{acq}(X), \texttt{rel}(X), \texttt{rel}(Y)$ of t.
4. We execute all t_y^2 (i.e., the second parts of all threads that correspond to the vectors in A_2) arbitrarily, but after the segment $\texttt{acq}(X), \texttt{rel}(X), \texttt{rel}(Y)$ of t.

We refer to the full paper [22] for the correctness of the reduction and thus the proof of Theorem 5.

5 Violations of the Locking Discipline

5.1 Lock-Cover Races

We start with a simple reduction from OV to detecting lock-cover races. Given a OV instance OV(n,d) on two vector sets A_1, A_2, we create a trace σ as follows. We have a single variable x and two threads t_1, t_2. We associate with each vector of the set A_i a write access event $e = \langle t_i, \texttt{w}(x) \rangle$. Moreover, each such event holds up to d locks, so that e holds the k^{th} lock iff k^{th} coordinate of the vector corresponding to the event is 1. The trace σ is formed by ordering the sequence of events corresponding to vectors of A_1 of OV first, in a fixed arbitrary order, followed by the sequence of events corresponding to A_2, again in arbitrary order. We refer to [22] for the correctness, which concludes the proof of Theorem 6.

5.2 Lock-Set Races

We now turn our attention to lock-set races. We first prove Theorem 9, i.e., that determining whether a trace σ has a lock-set race on a specific variable x can be performed in linear time.

A linear-time algorithm per variable. Verifying that there are two conflicting events on x is straightforward by a single pass of σ. The more involved part is in computing the lock-set of x, i.e., the set $\bigcap_{e \in \mathsf{Accesses}_\sigma(x)} \mathsf{locksHeld}_\sigma(e)$, in linear time. Indeed, each intersection alone requires $\Theta(\mathcal{L})$ time, resulting to $\Theta(\mathcal{N} \cdot \mathcal{L})$ time overall.

Here we show that a somewhat more involved algorithm achieves the task. The algorithm performs a single pass of σ, while maintaining three simple sets A, B, and C. While processing an event e, the sets are updated to maintain the invariant

$$A = \mathsf{locksHeld}_\sigma(e) \qquad B = \mathsf{Locks}_\sigma \cap \bigcap_{e' \in \mathsf{Accesses}_\sigma(x), e' \leq^\sigma_{\mathsf{tr}} e} \mathsf{locksHeld}_\sigma(e') \qquad C = \overline{A} \cap B \tag{2}$$

The sets are initialized as $A = \varnothing$, $B = C = \mathsf{Locks}_\sigma$. Then the algorithm performs a pass over σ and processes each event e according to the description of Algorithm 3.

Algorithm 3 *Computing the lock-set of variable x.*

```
acquire($t$, $\ell$):
    $A \leftarrow A \cup \{\ell\}$
    if $\ell \in B$ then
        $C \leftarrow C \setminus \{\ell\}$
release($t$, $\ell$):
    $A \leftarrow A \setminus \{\ell\}$
    if $\ell \in B$ then
        $C \leftarrow C \cup \{\ell\}$
read($t$, $y$):
    if $x = y$ then
        $B \leftarrow B \setminus C$
        $C \leftarrow \varnothing$
write($t$, $y$):
    if $x = y$ then
        $B \leftarrow B \setminus C$
        $C \leftarrow \varnothing$
```

The correctness of Algorithm 3 follows by proving the invariant in Equation (2). We refer to [22] for the details, which concludes the proof of Theorem 9.

Short witnesses for lock-set races. Besides the advantage of a faster algorithm, Theorem 9 implies that lock-set races have short witnesses that can be verified in linear time. This allows us to prove that detecting a lock-set race is in $\mathsf{NTIME}[\mathcal{N}] \cap \mathsf{coNTIME}[\mathcal{N}]$, and we can thus use [9, Corollary 2] to prove Theorem 7.

Proof of Theorem 7. First we argue that the problem is in $\mathsf{NTIME}[\mathcal{N}]$. Indeed, the certificate for the existence of a lock-set race is simply the variable x on which there is a lock-set race. By Theorem 9, verifying that we indeed have a lock-set race on x takes $O(\mathcal{N})$ time.

Now we argue that the problem is in $\mathsf{coNTIME}[\mathcal{N}]$, by giving a certificate to verify in linear time that σ does not have a race of the required form. The certificate has size $O(|\mathsf{Vars}_\sigma|)$, and specifies for every variable, either the lock that is held by all access events of the variable, or a claim that there exist no two conflicting events on that variable. The certificate can be easily verified by one pass over σ. ◀

Lock-set races are Hitting-Set hard. Finally we prove Theorem 8, i.e., that determining a single lock-set race is HS-hard, and thus also carries a conditional quadratic lower bound. We establish a fine-grained reduction from HS. Given a HS instance HS(n,d) on two vector sets

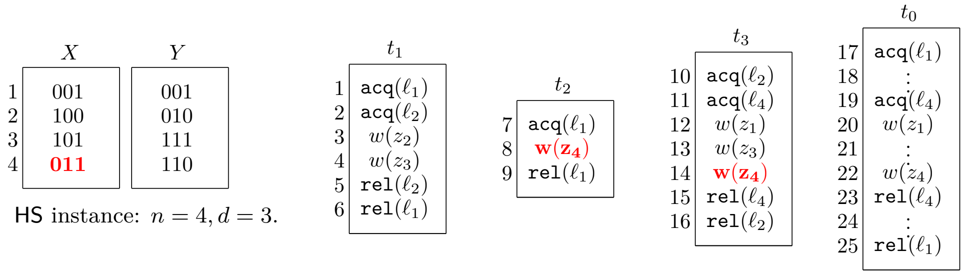


Figure 4 Reducing HS to detecting a lock-set race on trace σ with d threads. Thread t_k uses lock l_i if $y_i[k] = 0$, and $w(z_j)$ if $x_j[k] = 1$. Vector x_4 hits all vectors in Y, implying a lock-set race on z_4.

X, Y, we create a trace σ using $d+1$ threads $\{t_j\}_{j \in \{0\} \cup [d]}$, n locks $\{\ell_i\}_{i \in [n]}$, and n variables $\{z_i\}_{k \in [n]}$. Thread t_0 that executes $\texttt{acq}(\ell_1), \dots, \texttt{acq}(\ell_n), \texttt{w}(z_1), \dots \texttt{w}(z_n), \texttt{rel}(\ell_n), \dots \texttt{rel}(\ell_1)$. Each of the threads t_j, for $j \in [d]$, has a single nested critical section consisting of the locks $\ell_i \in [n]$ such that the i^{th} vector of Y has its j^{th} coordinate 0, i.e, $y_i[j] = 0$. The events in the critical section are all write events of all variables $z_k \in [n]$ with $x_k[j] = 1$. The trace orders all events of each thread t_d consecutively, and all the events overall in increasing order of d. See Figure 4 for an illustration. We refer to [22] for the correctness, which concludes the proof of Theorem 8.

6 Conclusion

In this work we have taken a fine-grained view of the complexity of popular notions of dynamic data races. We have established a range of lower bounds on the complexity of detecting HB races, sync-preserving races, as well as races based on the locking discipline (lock-cover/lock-set races). Moreover, we have characterized cases where lower bounds based on SETH are not possible under NSETH. Finally, we have proven new upper bounds for detecting HB and lock-set races. To our knowledge, this is the first work that characterizes the complexity of well-established dynamic race-detection techniques, allowing for a rigorous characterization of their trade-offs between expressiveness and running time.

References

1 Helgrind: a thread error detector. Accessed: 2021-04-30. URL: https://valgrind.org/docs/manual/hg-manual.html.

2 Intel inspector. Accessed: 2021-04-30. URL: https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/inspector.html.

3 Amir Abboud, Virginia Vassilevska Williams, and Joshua Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '16, page 377–391, USA, 2016. Society for Industrial and Applied Mathematics.

4 Utpal Banerjee, Brian Bliss, Zhiqiang Ma, and Paul Petersen. A theory of data race detection. In *Proceedings of the 2006 Workshop on Parallel and Distributed Systems: Testing and Debugging*, PADTAD '06, pages 69–78, New York, NY, USA, 2006. ACM. doi:10.1145/1147403.1147416.

5 Hans-J. Boehm. How to miscompile programs with "benign" data races. In *Proceedings of the 3rd USENIX Conference on Hot Topic in Parallelism*, HotPar'11, page 3, USA, 2011. USENIX Association.

6 Hans-J. Boehm. Position paper: Nondeterminism is unavoidable, but data races are pure evil. In *Proceedings of the 2012 ACM Workshop on Relaxing Synchronization for Multicore and Manycore Scalability*, RACES '12, page 9–14, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2414729.2414732.

7 Hans-J. Boehm and Sarita V. Adve. Foundations of the C++ Concurrency Memory Model. In *Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '08, page 68–78, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1375581.1375591.

8 Karl Bringmann. Fine-Grained Complexity Theory (Tutorial). In Rolf Niedermeier and Christophe Paul, editors, *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:7, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.STACS.2019.4.

9 Marco L Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 261–270, 2016.

10 Bernadette Charron-Bost. Concerning the size of logical clocks in distributed systems. *Information Processing Letters*, 39(1):11–16, 1991. doi:10.1016/0020-0190(91)90055-M.

11 Peter Chini, Jonathan Kolberg, Andreas Krebs, Roland Meyer, and Prakash Saivasan. On the Complexity of Bounded Context Switching. In Kirk Pruhs and Christian Sohler, editors, *25th Annual European Symposium on Algorithms (ESA 2017)*, volume 87 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 27:1–27:15, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2017.27.

12 Peter Chini, Roland Meyer, and Prakash Saivasan. Fine-grained complexity of safety verification. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 20–37, Cham, 2018. Springer International Publishing.

13 Peter Chini and Prakash Saivasan. A Framework for Consistency Algorithms. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020)*, volume 182 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 42:1–42:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.FSTTCS.2020.42.

14 Anne Dinning and Edith Schonberg. Detecting access anomalies in programs with critical sections. In *Proceedings of the 1991 ACM/ONR Workshop on Parallel and Distributed Debugging*, PADD '91, pages 85–96, New York, NY, USA, 1991. ACM. doi:10.1145/122759.122767.

15 Tayfun Elmas, Shaz Qadeer, and Serdar Tasiran. Goldilocks: A race and transaction-aware java runtime. In *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '07, pages 245–255, New York, NY, USA, 2007. ACM. doi:10.1145/1250734.1250762.

16 Cormac Flanagan and Stephen N. Freund. Fasttrack: Efficient and precise dynamic race detection. In *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '09, pages 121–133, New York, NY, USA, 2009. ACM. doi:10.1145/1542476.1542490.

17 Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. *ACM Trans. Algorithms*, 15(2), December 2018. doi:10.1145/3196275.

18 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

19 Ayal Itzkovitz, Assaf Schuster, and Oren Zeev-Ben-Mordehai. Toward integration of data race detection in dsm systems. *J. Parallel Distrib. Comput.*, 59(2):180–203, November 1999. doi:10.1006/jpdc.1999.1574.

20 Baris Kasikci, Cristian Zamfir, and George Candea. Racemob: Crowdsourced data race detection. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, SOSP '13, page 406–422, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2517349.2522736.

21 Dileep Kini, Umang Mathur, and Mahesh Viswanathan. Dynamic race prediction in linear time. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2017, pages 157–170, New York, NY, USA, 2017. ACM. doi:10.1145/3062341.3062374.

22 Rucha Kulkarni, Umang Mathur, and Andreas Pavlogiannis. Dynamic data-race detection through the fine-grained lens, 2021. arXiv:2107.03569.

23 Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978. doi:10.1145/359545.359563.

24 Umang Mathur, Dileep Kini, and Mahesh Viswanathan. What happens-after the first race? enhancing the predictive power of happens-before based dynamic race detection. *Proc. ACM Program. Lang.*, 2(OOPSLA):145:1–145:29, October 2018. doi:10.1145/3276515.

25 Umang Mathur, Andreas Pavlogiannis, and Mahesh Viswanathan. The complexity of dynamic data race prediction. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, page 713–727, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3373718.3394783.

26 Umang Mathur, Andreas Pavlogiannis, and Mahesh Viswanathan. Optimal prediction of synchronization-preserving races. *Proc. ACM Program. Lang.*, 5(POPL), January 2021. doi:10.1145/3434317.

27 Satish Narayanasamy, Zhenghao Wang, Jordan Tigani, Andrew Edwards, and Brad Calder. Automatically classifying benign and harmful data races using replay analysis. In *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '07, page 22–31, New York, NY, USA, 2007. Association for Computing Machinery. doi:10.1145/1250734.1250738.

28 Robert O'Callahan and Jong-Deok Choi. Hybrid dynamic data race detection. *SIGPLAN Not.*, 38(10):167–178, June 2003. doi:10.1145/966049.781528.

29 Andreas Pavlogiannis. Fast, sound, and effectively complete dynamic race prediction. *Proc. ACM Program. Lang.*, 4(POPL), 2019. doi:10.1145/3371085.

30 Eli Pozniansky and Assaf Schuster. Efficient on-the-fly data race detection in multithreaded C++ programs. *SIGPLAN Not.*, 38(10):179–190, 2003. doi:10.1145/966049.781529.

31 Jake Roemer, Kaan Genç, and Michael D. Bond. High-coverage, unbounded sound predictive race detection. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2018, pages 374–389, New York, NY, USA, 2018. ACM. doi:10.1145/3192366.3192385.

32 Grigore Rosu. Rv-predict, runtime verification, 2018. URL: https://runtimeverification.com/predict.

33 Mahmoud Said, Chao Wang, Zijiang Yang, and Karem Sakallah. Generating data race witnesses by an smt-based analysis. In *Proceedings of the Third International Conference on NASA Formal Methods*, NFM'11, pages 313–327, Berlin, Heidelberg, 2011. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=1986308.1986334.

34 Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, and Thomas Anderson. Eraser: A dynamic data race detector for multithreaded programs. *ACM Trans. Comput. Syst.*, 15(4):391–411, 1997. doi:10.1145/265924.265927.

35 Koushik Sen, Grigore Roşu, and Gul Agha. Detecting errors in multithreaded programs by generalized predictive analysis of executions. In Martin Steffen and Gianluigi Zavattaro, editors, *Formal Methods for Open Object-Based Distributed Systems*, pages 211–226, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

36 Konstantin Serebryany and Timur Iskhodzhanov. ThreadSanitizer: Data Race Detection in Practice. In *WBIA '09: Proceedings of the Workshop on Binary Instrumentation and Applications*, 2009.

37 Jaroslav Ševčík and David Aspinall. On validity of program transformations in the java memory model. In Jan Vitek, editor, *ECOOP 2008 – Object-Oriented Programming*, pages 27–51, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

38 Yannis Smaragdakis, Jacob Evans, Caitlin Sadowski, Jaeheon Yi, and Cormac Flanagan. Sound predictive race detection in polynomial time. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '12, pages 387–400, New York, NY, USA, 2012. ACM. doi:10.1145/2103656.2103702.

39 Martin Sulzmann and Kai Stadtmüller. Efficient, near complete, and often sound hybrid dynamic data race prediction. In *Proceedings of the 17th International Conference on Managed Programming Languages and Runtimes*, MPLR 2020, page 30–51, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3426182.3426185.

40 Christoph von Praun. *Race Detection Techniques*, pages 1697–1706. Springer US, Boston, MA, 2011. doi:10.1007/978-0-387-09766-4_38.

41 Jaroslav Ševčík. Safe optimisations for shared-memory concurrent programs. *SIGPLAN Not.*, 46(6):306–316, June 2011. doi:10.1145/1993316.1993534.

42 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2-3):357–365, 2005.

43 Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, volume 3, pages 3431–3472. World Scientific, 2018.

44 M. Zhivich and R. K. Cunningham. The real cost of software errors. *IEEE Security and Privacy*, 7(2):87–90, March 2009. doi:10.1109/MSP.2009.56.

A Fine-grained Complexity

Fine-grained Reductions. Assume that A and B are computational problems and $a(n)$ and $b(n)$ are their conjectured running time lower bounds, respectively. Then we say A (a, b)-reduces to B, denoted by A $_{(\mathsf{a})}\preceq_{(\mathsf{b})}$ B, if for every $\epsilon > 0$, there exists $\delta > 0$, and an algorithm R for A that runs in time $a(n)^{(1-\delta)}$ on inputs of length n, making q calls to an oracle for B with query lengths $n_1, \dots, n_q$, where, $\sum_1^q (b(n))^{(1-\epsilon)} \leq (a(n))^{(1-\delta)}$.

▶ **Hypothesis 3** (Strong Exponential Time Hypothesis (SETH)). *For every $\epsilon > 0$ there exists an integer $k \geq 3$ such that CNF-SAT on formulas with clause size at most k and n variables cannot be solved in $O(2^{(1-\epsilon)n})$ time even by a randomized algorithm.*

▶ **Hypothesis 4** (Non-deterministic SETH (NSETH)). *For every $\epsilon > 0$, there exists a k so that k-TAUT is not in $\mathsf{NTIME}[2^{\mathsf{n}(1-\epsilon)}]$, where k-TAUT is the language of all k-DNF formulas which are tautologies.*

B Proofs of Section 3

B.1 Proofs from Section 3.1

▶ **Lemma 11.** *Let $e_1 \leq^\sigma_{tr} e_2$ be events in σ such that $\mathsf{tid}(e_1) \neq \mathsf{tid}(e_2)$. We have, $e_1 \leq^\sigma_{HB} e_2 \iff \neg(\mathsf{AcqLS}^\sigma_{e_2} \sqsubseteq \mathsf{RelLS}^\sigma_{e_1})$*

Proof. ($\Rightarrow$) Let $e_1 \leq^\sigma_{\mathsf{HB}} e_2$. Using the definition of $\leq^\sigma_{\mathsf{HB}}$, there must be a sequence of events $f_1, f_2 \dots f_k$ with $k > 1$, $f_1 = e_1$, $f_k = e_2$, and for every $1 \leq i < k$, $f_i \leq^\sigma_{\mathsf{tr}} f_{i+1}$ and either $f_i \leq^\sigma_{\mathsf{TO}} f_{i+1}$ or there is a lock ℓ, such that $f_i \in \mathsf{Releases}_\sigma(\ell)$ and $f_{i+1} \in \mathsf{Acquires}_\sigma(\ell)$. Let j be the smallest index i such that $\mathsf{tid}(f_i) \neq \mathsf{tid}(f_{i+1})$; such an index exists as $\mathsf{tid}(e_1) \neq \mathsf{tid}(e_2)$. Observe that there must be a lock ℓ for which $\mathrm{op}(f_j) = \mathtt{rel}(\ell)$ and $\mathrm{op}(f_{j+1}) = \mathtt{acq}(\ell)$. Observe that $pos_\sigma(f_j) < pos_\sigma(f_{j+1})$, $\mathsf{RelLS}^\sigma_{e_1}(\ell) \leq pos_\sigma(f_j)$ and $pos_\sigma(f_{j+1}) \leq \mathsf{AcqLS}^\sigma_{e_2}$, giving us $\mathsf{RelLS}^\sigma_{e_1}(\ell) < \mathsf{AcqLS}^\sigma_{e_2}(\ell)$.

($\Leftarrow$) Let ℓ be a lock such that $\mathsf{RelLS}^\sigma_{e_1}(\ell) < \mathsf{AcqLS}^\sigma_{e_2}(\ell)$. Then, there is a release event f and an acquire event g on lock ℓ such that $pos_\sigma(f) < pos_\sigma(g)$,$e_1 \leq^\sigma_{\mathsf{HB}} f$ and $g \leq^\sigma_{\mathsf{HB}} e_2$. This means $f \leq^\sigma_{\mathsf{HB}} g$ and thus $e_1 \leq^\sigma_{\mathsf{HB}} e_2$. ◀

For the sake of completeness, we present the computation of release lockstamps. The computation of release lockstamps takes place in the reverse order of $\leq^{\sigma}_{\mathsf{tr}}$ (i.e., from right to left), unlike the case of acquire lockstamps. As with Algorithm 1, we maintain the following variables. For each thread t and lock ℓ, we will maintain variables $\mathbb{C}_t$ and $\mathbb{L}_\ell$ that take values from the space of all lockstamps. We also additionally maintain an integer variable p_ℓ for each lock ℓ that stores the index (or relative position) of the earliest (according to the trace order $\leq^{\sigma}_{\mathsf{tr}}$) release event of lock ℓ in the trace suffix seen so far. Initially, we set $\mathbb{C}_t$ and $\mathbb{L}_m$ to $\lambda\ell \cdot \infty$, for each thread t and lock m. Further, for each lock m, we set p_m to $n_m + 1$, where n_m is the number of release events of m in the trace; this can be obtained in a linear scan (or by reading the value of p_m at the end of a run of Algorithm 1). We traverse the events in reverse, and perform updates to the data structures as described in Algorithm 4, by invoking the appropriate *handler* based on the thread and operation of the event $e = \langle t, op\rangle$ being visited. At the end of each handler, we assign the lockstamp $\mathsf{RelLS}^{\sigma}_{e}$ to the event e.

Algorithm 4 *Assigning release lockstamps to events in the trace. Events are processed in reverse order.*

```
acquire(t, ℓ):
    $\mathbb{L}_\ell \leftarrow \mathbb{C}_t$
    $\mathsf{RelLS}^{\sigma}_{e} \leftarrow \mathbb{C}_t$

release(t, ℓ):
    $\mathsf{p}_\ell \leftarrow \mathsf{p}_\ell - 1$
    $\mathbb{C}_t \leftarrow \mathbb{C}_t[\ell \mapsto \mathsf{p}_\ell] \sqcap \mathbb{L}_\ell$
    $\mathsf{RelLS}^{\sigma}_{e} \leftarrow \mathbb{C}_t$

read(t, x):
    $\mathsf{RelLS}^{\sigma}_{e} \leftarrow \mathbb{C}_t$

write(t, x):
    $\mathsf{RelLS}^{\sigma}_{e} \leftarrow \mathbb{C}_t$
```

Let us now state the correctness of Algorithm 1 and Algorithm 4.

▶ **Lemma 14.** *On input trace σ, Algorithm 1 and Algorithm 4 correctly compute the lockstamps $\mathsf{AcqLS}^{\sigma}_{e}$ and $\mathsf{RelLS}^{\sigma}_{e}$ respectively for each event $e \in \mathsf{Events}_\sigma$.*

Proof Sketch. We focus on the correctness proof of Algorithm 1; the proof for Algorithm 4 is similar. The proof relies on the invariant maintained by Algorithm 1 the variables $\mathbb{C}_t$, $\mathbb{L}_\ell$ and p_ℓ for each thread t and lock ℓ, which we state next. Let π be the prefix of the trace processed at any point in the algorithm. Let C^{π}_{t}, L^{π}_{ℓ} and p^{π}_{ℓ} be the values of the variables $\mathbb{C}_t$, $\mathbb{L}_\ell$ and p_ℓ after processing the prefix π. Then, the following invariants are true:

- $C^{\pi}_{t} = \mathsf{AcqLS}^{\pi}_{e^{\pi}_{t}} = \mathsf{AcqLS}^{\sigma}_{e^{\pi}_{t}}$, where e^{π}_{t} is the last event in π performed by thread t
- $L^{\pi}_{\ell} = \mathsf{AcqLS}^{\pi}_{e^{\pi}_{\ell}} = \mathsf{AcqLS}^{\sigma}_{e^{\pi}_{\ell}}$, where e^{π}_{ℓ} is the last acquire event on lock ℓ in π.
- $p^{\pi}_{\ell} = pos^{\pi}_{\ell}(e^{\pi}_{\ell})$, where e^{π}_{ℓ} is the last acquire event on lock ℓ in π.

These invariants can be proved using a straightforward induction on the length of the trace, each time noting the definition of $\leq^{\sigma}_{\mathsf{HB}}$. ◀

▶ **Lemma 15.** *For a trace with $\mathcal{N}$ events and $\mathcal{L}$ locks, Algorithm 1 and Algorithm 4 both take $O(\mathcal{T} \cdot \mathcal{L})$ time.*

Proof. We focus on Algorithm 1; the analysis for Algorithm 4 is similar. At each acquire event, the algorithm spends $O(1)$ time for updating p_ℓ, $O(\mathcal{L})$ time for doing the $\sqcup$ operation, and $O(\mathcal{L})$ time for the copy operation ('$\mathsf{AcqLS}^{\sigma}_{e} \leftarrow \mathbb{C}_t$'). For a release event, we spend $O(\mathcal{L})$ for the two copy operations. At read and write events, we spend $O(\mathcal{L})$ for copy operations. This gives a total time of $O(\mathcal{N} \cdot \mathcal{L})$. ◀

▶ **Lemma 12.** *A trace σ has an HB-race iff there is pair of consecutive conflicting events in σ that is an HB-race. Moreover, σ has at most $O(\mathcal{N})$ many consecutive conflicting pairs of events.*

Proof. We first prove that if there is a an HB-race in σ, then there is a pair of consecutive conflicting events that is in HB-race. Consider the first HB-race, i.e., an HB-race (e_1, e_2) such that for every other HB-race (e'_1, e'_2), either $e_2 \leq^{\sigma}_{\mathsf{tr}} e'_2$ or $e_2 = e'_2$ and $e'_1 \leq^{\sigma}_{\mathsf{tr}} e_1$. We remark that such a race (e_1, e_2) exists if σ has any HB-race. We now show that (e_1, e_2) are a consecutive conflicting pair (on variable x). Assume on the contrary that there is an event $f \in \mathsf{Writes}_{\sigma}(x)$ such that $e_1 <^{\sigma}_{\mathsf{tr}} f <^{\sigma}_{\mathsf{tr}} e_2$. If either (e_1, f) or (f, e_2) is an HB-race, then this contradicts our assumption that (e_1, e_2) is the first HB-race in σ. Thus, $e_1 \leq^{\sigma}_{\mathsf{HB}} f$ and $f \leq^{\sigma}_{\mathsf{HB}} e_2$, which gives $e_1 \leq^{\sigma}_{\mathsf{HB}} e_2$, another contradiction.

We now turn our attention to the number of consecutive conflicting events in σ. For every read or write event e_2, there is at most one write event e_1 such that (e_1, e_2) is a consecutive conflicting pair (namely the latest conflicting write event before e_2). Further, for every read event e_1, there is at most one write event e_2 such that (e_1, e_2) is a consecutive conflicting pair (namely the earliest conflicting write event after e_1). This gives at most $2\mathcal{N}$ consecutive conflicting pairs of events. ◀

Let us now state the correctness of Algorithm 2.

▶ **Lemma 16.** *For a trace σ, Algorithm 2 reports a race iff σ has an HB-race.*

Proof Sketch. The proof relies on the following straightforward invariants; we skip their proofs as they are straightforward. In the following, e^{π}_x is the last event with $\mathrm{op}(e^{\pi}_x) = \mathtt{w}(x)$ in a trace π.

- After processing the prefix π of σ, $\mathsf{t}^w_x = \mathsf{tid}(e^{\pi}_x)$ and $\mathbb{W}_x = \mathsf{RelLS}^{\sigma}_{e^{\pi}_x}$.
- After processing the prefix π of σ, the set S_x is $\{(\mathsf{tid}(e), \mathsf{RelLS}^{\sigma}_e) \mid e \in \mathsf{Reads}_{\pi}(x), e^{\pi}_x \leq^{\pi}_{\mathsf{tr}} e\}$.

The rest of the proof follows from Lemma 11 and Lemma 12. ◀

Let us now characterize the time complexity of Algorithm 2.

▶ **Lemma 17.** *On an input trace with $\mathcal{N}$ events and $\mathcal{L}$ locks, Algorithm 2 runs in time $O(\mathcal{N} \cdot \mathcal{L})$.*

Proof Sketch. Each pair (t, L) of thread identifier and lockstamp is added atmost once in some set S_x (for some x). Also, each such pair is also compared against another timestamp atmost once. Each comparison of timestamps take $O(\mathcal{L})$ time. This gives a total time of $O(\mathcal{N} \cdot \mathcal{L})$. ◀

▶ **Theorem 4.** *Deciding whether σ has an HB race can be done in time $O(\mathcal{N} \cdot \min(\mathcal{T}, \mathcal{L}))$.*

Proof. We focus on proving that there is an $O(\mathcal{N} \cdot \mathcal{L})$ time algorithm, as the standard vector-clock algorithm [19] for checking for an HB-race runs in $O(\mathcal{N} \cdot \mathcal{T})$ time. Our algorithm's correctness is stated in Lemma 14 and Lemma 16 and its total running time is $O(\mathcal{N} \cdot \mathcal{L})$ (Lemma 17 and Lemma 15). ◀

B.2 Proofs from Section 3.2

▶ **Theorem 1.** *For any $\epsilon > 0$, there is no algorithm that detects even a single HB race that involves a read in time $O(\mathcal{N}^{2-\epsilon})$, unless the OV hypothesis fails.*

Proof. Consider a pair of events $\mathtt{w}(z)$ from the d threads $t(x, i), i \in [d]$, and $\mathtt{r}(z) \in t_y$ for some x, i, y. We have $\mathtt{w}(z) \leq^{\sigma}_{\mathsf{HB}} \mathtt{r}(z)$ iff there is some path from $\mathtt{w}(z)$ to $\mathtt{r}(z)$ in $\mathsf{G}^{\sigma}_{\mathsf{HB}}$. As $\mathtt{w}(z)$ and $\mathtt{r}(z)$ are in different threads, such a path can only be through lock events in a sequence of threads such that the first and last threads are $t(x, i)$ for some $i \in [d]$ and t_y, and every consecutive pair of threads in the sequence holds a common lock. Now all the locks in t_y are $l(y, i)$ for all i where $y[i] = 1$. Consider the lock corresponding to any $i \in [d]$. The only thread $t(x', i)$ that also holds this lock corresponds to the last x' such that $x'[i] = 1$. The only other lock held by $t(x', i)$ is l_i. If $\mathtt{w}(z)$ is in $t(x', i)$, we are done. Otherwise the only common lock between these threads $t(x', i)$ and those of $\mathtt{w}(z)$ can be one of the l_i. The threads of $\mathtt{w}(z)$ contain all l_i where $x[i] = 1$. Hence, for there to be a common lock between these threads, there must be at least one i such that $x'[i] = 1$ and $x[i] = 1$. As this thread also has the lock $l(y, i)$, $y[i]$ is also 1.

Thus, there is a path from $\mathtt{w}(z)$ to $\mathtt{r}(z)$ if and only if there is at least one $i \in [d]$ such that $x[i] = y[i] = 1$, hence x and y are not orthogonal. A pair of orthogonal vectors of OV thus corresponds to a write-read HB-race in the reduced trace.

Finally we turn our attention to the complexity. In time $O(n \cdot d)$, we have reduced an OV instance to determining whether there is a write-read HB race in a trace of $\mathcal{N} = O(nd)$ events. If there was a sub-quadratic i.e. $O((n \cdot d)^{(2-\epsilon)}) = n^{(2-\epsilon)} \cdot \text{poly}(d)$ algorithm for detecting a write-read HB race, then this would also solve OV in $n^{(2-\epsilon)} \cdot \text{poly}(d)$ time, refuting the OV hypothesis. ◀

▶ **Lemma 13.** *FO(∀∃∃) reduces to MCONN on a graph G with $O(n)$ nodes in $O(n^2)$ time.*

Proof. For intuition, assume the first order property is on an undirected graph with n variables and m edges. Let the property be specified in quantified 3-DNF form with a constant number of predicates, i.e., $\phi = \forall x \exists y \exists z \ (\psi_1 \vee \psi_2 \vee \ldots \psi_k)$, where x, y, z represent nodes of the graph, and each ψ_i is a conjunction of 3 variables representing edges of the graph, for example $e(x, y) \wedge \neg e(y, z) \wedge e(x, z)$. The property is then true if and only if some predicate is satisfied, which is true if all of its variables are satisfied ($e(x, y)$ is satisfied when edge (x, y) is in the graph). Denote the graph on which ϕ is defined by $H(I, J)$, where I and J are respectively the sets of nodes and edges of H.

The instance of MCONN is constructed given H and ϕ as follows. Construct a $(2k + 2)$-partite graph $G(V, E)$ by first creating $2k+2$ copies of I. Denote these copies by $S, Y_i, Z_i, T,\ i \in [k]$, and the copy of each node $x \in I$ in any part, say S, by $x(S)$. $\psi_i = (e_1 \wedge e_2 \wedge e_3)$ is encoded by connecting the sets (S, Y_i) to represent e_1, (Y_i, Z_i) for e_2 and (Z_i, T) for e_3 as follows. If e_i is of the form $e(x, y)$ (and not its negation), then draw a copy of H between its corresponding sets, say S and Y_i without loss of generality. That is, for every x, y, $(x, y) \in J \Leftrightarrow (x(S), y(Y_i)) \in E$. If on the other hand e_i is of the form $\neg e(x, y)$ then connect a copy of the complement of H, i.e., $(x, y) \notin J \Leftrightarrow (x(S), y(Y_i)) \in E$.

Finally define $|I|$ pairs $(x(S), x(T))$ as the (s, t) pairs for MCONN.

We now prove this reduction is correct. First, assume ϕ is true. Then for every node x, there exist nodes y, z such that some predicate is true. If ψ_i is the predicate that is satisfied for some node u, then there is a path between $u(S)$ and $u(T)$ through the parts S, Y_i, Z_i and T as follows. As the first variable is satisfied, then if it is $e(x, y)$, then $(x, y) \in J$, and $x(S)$ is connected to $y(Y_i)$, and if it is $\neg e(x, y)$, then $(x, y) \notin J$ and again $x(S)$ is connected to $y(Y_i)$. Similarly, $y(Y_i)$ is connected to $z(Z_i)$, and $z(Z_i)$ to $x(T)$. These edges form a 3 length path between $x(S)$ and $x(T)$.

Now consider the reverse case, and assume the MCONN problem is true, that is , there is a path between every $(x(S), x(T))$ pair. Note that the construction of edges in G is such that any path from $x(S)$ to $x(T)$ has to be a 3 length path, connecting the copy of x in S to its copy in some Y_i, from this Y_i to its corresponding Z_i, and from Z_i to T. Also, this path exists only if all variables of the corresponding ψ_i are true. Hence, as there is a path between every pair $(x(S), x(T))$, and one pair is defined for every variable x, some predicate is satisfied for every x. Thus ϕ is also true.

Finally, the time of the reduction is equal to the size of G. This is $2k + 2 = O(1)$ graphs, each of which is either H or its complement. Hence $|G| = O(m+n+(n^2-m)+n) = O(n^2)$. ◀

▶ **Theorem 3.** *For any $\epsilon > 0$, if there is an algorithm for detecting any* HB *race in time $O(\mathcal{N}^{1+\epsilon})$, then there is an algorithm for* FO(∀∃∃) *formulas in time $O(m^{1+\epsilon})$.*

Proof. We first reduce the instance of FO(∀∃∃) to MCONN as in the proof of Lemma 13. Let $G(V, E)$ be the multi-partite graph for MCONN and S, T the first and last parts of nodes of G. We add a sufficient number of nodes, referred as dummy nodes, to make G sparse. Let every node x of $V \backslash T$ correspond to a distinct thread t_x and form one write access event to a distinct variable v_x in the thread. Let each node t in T also correspond to a write access event of the variable corresponding to the copy of t in S, and be in a new thread. Define $|E|$ locks, and for every edge $(a, b) \in E$, let the events corresponding to v_a and v_b hold the lock $l_{(a,b)}$ corresponding to (a, b). The trace σ for first lists all threads corresponding to the dummy nodes in some fixed arbitrary order, then the threads corresponding to nodes in S, followed by those in each Y_i, followed by those in each Z_i, in a fixed arbitrary order, and finally those in T.

This reduction is seen to be correct by observing that G was modified to be the transitive reduction graph of σ, and the only HB-race events can be the pairs of write events corresponding to the pairs of nodes given as input to MCONN. Thus, each pair of events does not form an HB-race if and only if G has a path between its corresponding pair of nodes.

To analyze the time of the reduction, first we see that the size of σ is the size of G, with dummy nodes added to have $n = O(n^2)$, and hence $O(n^2)$. There are $O(n^2)$ variables, locks and threads in σ. If deciding if the given trace has an HB-race has an $O((n^2)^{1+\epsilon})$ time algorithm, then FO(∀∃∃) can be solved in $O(n^{2+\epsilon'})$ time, which is $O(m^{1+\epsilon'})$ time for properties on dense structures. ◀

Due to space constraints, we include the remaining proofs of the Theorems from Sections 4 and 5 in the full paper [22].

Adaptive Synchronisation of Pushdown Automata

A. R. Balasubramanian
Technische Universität München, Germany

K. S. Thejaswini
Department of Computer Science, University of Warwick, Coventry, UK

Abstract

We introduce the notion of adaptive synchronisation for pushdown automata, in which there is an external observer who has no knowledge about the current state of the pushdown automaton, but can observe the contents of the stack. The observer would then like to decide if it is possible to bring the automaton from any state into some predetermined state by giving inputs to it in an *adaptive* manner, i.e., the next input letter to be given can depend on how the contents of the stack changed after the current input letter. We show that for non-deterministic pushdown automata, this problem is 2-EXPTIME-complete and for deterministic pushdown automata, we show EXPTIME-completeness.

To prove the lower bounds, we first introduce (different variants of) subset-synchronisation and show that these problems are polynomial-time equivalent with the adaptive synchronisation problem. We then prove hardness results for the subset-synchronisation problems. For proving the upper bounds, we consider the problem of deciding if a given alternating pushdown system has an accepting run with at most k leaves and we provide an $n^{O(k^2)}$ time algorithm for this problem.

2012 ACM Subject Classification Theory of computation → Grammars and context-free languages; Theory of computation → Problems, reductions and completeness

Keywords and phrases Adaptive synchronisation, Pushdown automata, Alternating pushdown systems

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.17

Related Version *Full Version*: `https://arxiv.org/abs/2102.06897`

Funding *A. R. Balasubramanian*: Supported by funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 787367 (PaVeS).
K. S. Thejaswini: Supported by the EPSRC grant EP/P020992/1 (Solving Parity Games in Theory and Practice).

Acknowledgements We would like to thank Dmitry Chistikov for referring us to previous works on this topic.

1 Introduction

The notion of a synchronizing word for finite-state machines is a classical concept in computer science which consists of deciding, given a finite-state machine, whether there is a word which brings all of its states to a single state. Intuitively, assuming that we initially do not know which state the machine is in, such a word *synchronises* it to a single state and assists in regaining control over the machine.

This idea has been studied for many types of finite-state machines [24, 22, 2, 9] with applications in biocomputing [3], planning and robotics [10, 19] and testing of reactive systems [18, 14]. In recent years, the notion of a synchronizing word has been extended to various *infinite-state systems* such as timed automata [8], register automata [20], nested word automata [7], pushdown and visibly pushdown automata [11, 12]. In particular, for the pushdown case, Fernau, Wolf and Yamakami [12] have shown that this problem is undecidable even for deterministic pushdown automata.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 17; pp. 17:1–17:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

When the finite-state machine can produce outputs, the notion of synchronisation has been further refined to give rise to synchronisation under *partial observation* or *adaptive synchronisation* (See Chapter 1 of [5] and [17]). In this setting, there is an external observer who does not know the current state of the machine, however she can give inputs to the machine and observe the outputs given by the machine. Depending on the outputs of the machine, she can *adaptively* decide which input letter to give next. In this manner, the observer would like to bring the machine into some predetermined state. Larsen, Laursen and Srba [17] describe an example of adaptive synchronisation pertaining to the orientation of a simplified model of satellites, in which they observe that adaptively choosing the input letter is sometimes necessary in order to achieve synchronisation. In this paper, we extend this notion of adaptive synchronisation to pushdown automata (PDA). In our model, the observer does not know which state the PDA is currently in, but can observe the contents of the stack. She would then like to decide if it is possible to synchronise the PDA into some state by giving inputs to the PDA adaptively, i.e., depending on how the stack changes after each input. To the best of our knowledge, the notion of adaptive synchronisation has not been considered before for any class of infinite-state systems.

This question is a natural extension of the notion of adaptive synchronisation from finite-state machines to pushdown automata. Further, it is mentioned in the works of Lakhotia, Uday Kumar and Venable as well as Song and Touili [21, 16] that several antivirus systems determine whether a program is malicious by observing the calls that the program makes to the operating system. With this in mind, Song and Touili use pushdown automata [21] as abstractions of programs where a stack stores the calls made by the program and use this abstraction to detect viruses. Hence, we believe that our setting of being able to observe the changes happening to the stack can be practically motivated.

Our main results regarding adaptive synchronisation are as follows: We show that for non-deterministic pushdown automata, the problem is 2-EXPTIME-complete. However, by restricting our input to deterministic pushdown automata, we show that we can get EXPTIME-completeness, thereby obtaining an exponential reduction in complexity.

We also consider a natural variant of this problem, called *subset adaptive synchronisation*, which is similar to adaptive synchronisation, except the observer has more knowledge about which state the automaton is initially in. We obtain a surprising result that shows that this variant is polynomial-time equivalent to adaptive synchronisation, unlike in the case of finite-state machines. Furthermore, for the deterministic case of this variant, we obtain an algorithm that runs in time $O\left(n^{ck^3}\right)$ where n is the size of the input and k is the size of the subset of states that the observer believes the automaton is initially in. This gives a polynomial time algorithm if k is fixed and a quasi-polynomial time algorithm if $k = O(\log n)$.

Used as a subroutine in the above decision procedure, is an $O\left(n^{ck^2}\right)$ time algorithm to the following question, which we call the *sparse-emptiness problem*: Given an alternating pushdown system and a number k, decide whether there is an accepting run of the system with at most k leaves. Intuitively, such a run means that the system has an accepting run in which it uses only "limited universal branching". We note that such a notion of alternation with "limited universal branching" has recently been studied by Keeler and Salomaa for alternating finite-state automata [15]. Our problem can be considered as a generalisation of one of their problems (Corollary 2 of [15]) to pushdown systems. We think that this problem and its associated algorithm might be of independent interest.

Roadmap. In Section 2, we introduce notations. In Section 3, we discuss different variations of the problem. In Sections 4 and 5 we prove lower and upper bounds respectively. Proofs of some of our technical results can be found in the long version of this extended abstract available on arXiv [1].

2 Preliminaries

Given a finite set X, we let X^* denote the set of all words over the alphabet X. As usual, the concatenation of two words $x, y \in X^*$ is denoted by xy.

2.1 Pushdown Automata

We recall the well-known notion of a pushdown automaton. A pushdown automaton (PDA) is a 4-tuple $\mathcal{P} = (Q, \Sigma, \Gamma, \delta)$ where Q is a finite set of *states*, Σ is the *input alphabet*, Γ is the *stack alphabet* and $\delta \subseteq (Q \times \Sigma \times \Gamma) \times (Q \times \Gamma^*)$ is the *transition relation*. Alternatively, sometimes we will describe the transition relation δ as a function $Q \times \Sigma \times \Gamma \mapsto 2^{Q \times \Gamma^*}$. We will always use small letters $a, b, c, \ldots$ to denote elements of Σ, capital letters $A, B, C, \ldots$ to denote elements of Γ and Greek letters $\gamma, \eta, \omega, \ldots$ to denote elements of Γ^*.

If $(p, a, A, q, \gamma) \in \delta$ then we sometimes denote it by $(p, A) \xhookrightarrow{a} (q, \gamma)$. We say A is the *top* of the stack that is *popped* and γ is the string that is *pushed* onto the stack. A *configuration* of the automaton is a tuple (q, γ) where $q \in Q$ and $\gamma \in \Gamma^*$. Given two configurations $(q, A\gamma)$ and $(q', \gamma'\gamma)$ of $\mathcal{P}$ with $A \in \Gamma$, we say that $(q, A\gamma) \xrightarrow{a} (q', \gamma'\gamma)$ iff $(q, A) \xhookrightarrow{a} (q', \gamma')$.

As is usual, we assume that there exists a special *bottom-of-the-stack* symbol $\perp \in \Gamma$, such that whenever some transition pops $\perp$, it pushes it back in the bottom-most position. A PDA is said to be deterministic if for every $q \in Q$, $a \in \Sigma$ and $A \in \Gamma$, $\delta(q, a, A)$ has exactly one element. If a PDA is deterministic, we further abuse notation and denote $\delta(q, a, A)$ as a single element and not as a set.

2.2 Adaptive Synchronisation

We first expand upon the intuition given in the introduction for adaptive synchronisation with the help of a running example. Consider the pushdown automaton as given in Figure 1 where we do not know which state the automaton is in currently, but we do know that the stack content is $\perp$. To synchronise the automaton to the state 4 when the stack is visible, the observer has a strategy as depicted in Figure 2. The labelling of the nodes of the tree intuitively denotes the "knowledge of the observer" at the current point in the strategy and the labelling of the edges denotes the letter that she inputs to the PDA. Initially, according to the observer, the automaton could be in any one of the 4 states. The observer first inputs the letter $\square$. If the top of the stack becomes •, then she knows that the automaton is currently either in state 1 or 2. On the other hand, if the top of the stack becomes •, then the observer can deduce that the automaton is currently in state 3 or 4. From these two scenarios, by following the appropriate strategy depicted in the figure, we can see that she can synchronise the automaton to state 4. However, if the stack was hidden to the observer, reading either $\diamond$ or $\square$ does not change the knowledge of the observer and therefore, there is no word that can be read that would synchronise the automaton to any state.

We now formalize the notion of an adaptive synchronizing word that we have so far described. Let $\mathcal{P} = (Q, \Sigma, \Gamma, \delta)$ be a PDA. Given $S \subseteq Q$, $a \in \Sigma$ and $A \in \Gamma$, let $T^a_{S,A} := \{t \in \delta \mid t = (p, a, A, q, \gamma) \text{ where } p \in S\}$. Intuitively, if the observer knows that $\mathcal{P}$ is currently in some state in S and the top of the stack is A and she chooses to input a, then $T^a_{S,A}$ is

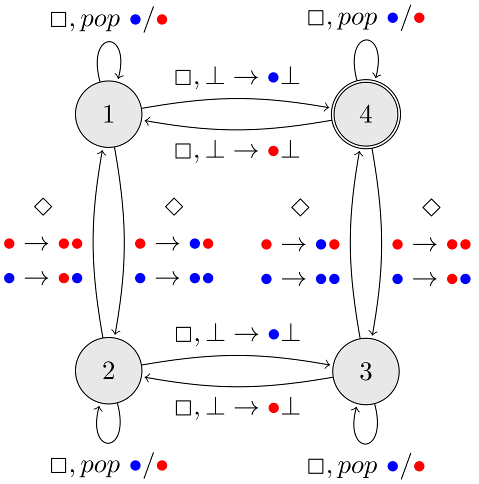


Figure 1 A label of the form $a, A \to \gamma$ means that if the input is a and if the top of the stack is A, then pop A and push γ.

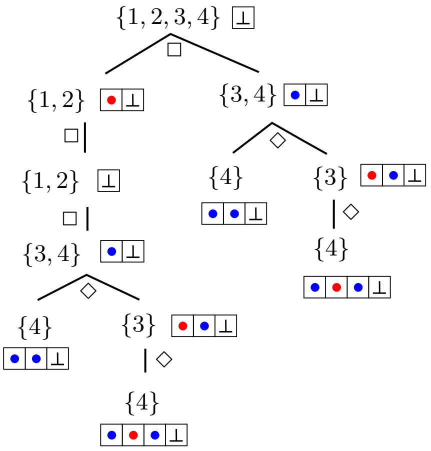


Figure 2 A synchroniser between $(\{1,2,3,4\}, \perp)$ and state 4 for the PDA in Figure 1.

the set of transitions that might take place. We define an equivalence relation $\sim^a_{S,A}$ on the elements of $T^a_{S,A}$ as follows: $t_1 \sim^a_{S,A} t_2 \iff \exists \gamma \in \Gamma^*$ such that $t_1 = (p_1, a, A, q_1, \gamma)$ and $t_2 = (p_2, a, A, q_2, \gamma)$. Notice that if $t_1 \sim^a_{S,A} t_2$ then the observer cannot distinguish occurrences of t_1 from occurrences of t_2. In our running example, if we take $S = \{3, 4\}$, $a = \Diamond$ and $A =$ 🔵, it is easy to see that $T^a_{S,A}$ is $\{(3, \Diamond,$ 🔵$, 4,$ 🔵🔵$), (4, \Diamond,$ 🔵$, 3,$ 🔴🔵$)\}$ and these two transitions are not in the same equivalence class under $\sim^a_{S,A}$.

The relation $\sim^a_{S,A}$ partitions the elements of $T^a_{S,A}$ into equivalence classes. If E is an equivalence class of $\sim^a_{S,A}$, then notice that there is a word $\gamma \in \Gamma^*$ such that all the transitions in E pop A and push γ onto the stack. This word γ will be denoted by $\mathit{word}(E)$. If we define $\mathit{next}(E) := \{q \mid (p, a, A, q, \mathit{word}(E)) \in E\}$, then $\mathit{next}(E)$ contains all the states that the automaton can move to if *any* of the transitions from E occur. Now, suppose the observer knows that $\mathcal{P}$ is currently in some state in S with A being at the top of the stack. Assuming she inputs the letter a and observes that A has been popped and $\mathit{word}(E)$ has been pushed, she can deduce that $\mathcal{P}$ is currently in some state in $\mathit{next}(E)$. In our running example of $S = \{3, 4\}$, $a = \Diamond$ and $A =$ 🔵, there are two equivalence classes $E_1 = \{(3, \Diamond,$ 🔵$, 4,$ 🔵🔵$)\}$ and $E_2 = \{(4, \Diamond,$ 🔵$, 3,$ 🔴🔵$)\}$ with $\mathit{next}(E_1) = \{3\}$, $\mathit{next}(E_2) = \{4\}$, $\mathit{word}(E_1) = \{$🔵🔵$\}$ and $\mathit{word}(E_2) = \{$🔴🔵$\}$.

A *pseudo-configuration* of the automaton $\mathcal{P}$ is a pair (S, γ) such that $S \subseteq Q$ and $\gamma \in \Gamma^*$. The pseudo-configuration (S, γ) captures the knowledge of the observer at any given point. Given a pseudo-configuration $(S, A\gamma)$ and an input letter a, let $\mathit{Succ}(S, A\gamma, a) := \{(\mathit{next}(E_1), \mathit{word}(E_1)\gamma), \ldots, (\mathit{next}(E_k), \mathit{word}(E_k)\gamma)\}$ where $E_1, \ldots, E_k$ are the equivalence classes of $\sim^a_{S,A}$. Each element of $\mathit{Succ}(S, A\gamma, a)$ will be called a possible successor of $(S, A\gamma)$ under the input letter a. The function Succ captures all the possible pseudo-configurations that could happen when the observer inputs a at the pseudo-configuration $(S, A\gamma)$.

We now define the notion of a *synchroniser* which will correspond to a strategy for the observer to synchronise the automaton into some state. Let $I \subseteq Q$, $s \in Q$ and $\gamma \in \Gamma^*$. (The I stands for ***I**nitial set of states*, and the s stands for ***s**ynchronising state*). A *synchroniser* between the pseudo-configuration (I, γ) and the state s, is a labelled tree T such that

- All the edges are labelled by some input letter $a \in \Sigma$ such that, for every vertex v, all its outgoing edges have the same label.

- The root is labelled by the pseudo-configuration (I, γ).
- Suppose v is a vertex which is labelled by the pseudo-configuration $(S, A\eta)$. Let a be the unique label of its outgoing edges and let $Succ(S, A\eta, a)$ be of size k. Then v has k children, with the i^{th} child labelled by the i^{th} pseudo-configuration in $Succ(S, A\eta, a)$.
- For every leaf, there exists $\eta \in \Gamma^*$ such that its label is $(\{s\}, \eta)$.

In addition, if all the leaves are labelled by $(\{s\}, \bot)$, then T is called a *super-synchroniser* between (I, γ) and s. We use the notation $(I, \gamma) \underset{\mathcal{P}}{\Rightarrow} s$ (resp. $(I, \gamma) \overset{\text{sup}}{\underset{\mathcal{P}}{\Longrightarrow}} s$) to denote that there is a synchroniser (resp. super-synchroniser) between (I, γ) and s in the PDA $\mathcal{P}$. (When $\mathcal{P}$ is clear from context, we drop it from the arrow notation).

2.3 Different Formulations

We now formally introduce the problem which we will refer to as the *adaptive synchronising problem* (Ada-Sync) and it is defined as the following:

Given: A PDA $\mathcal{P} = (Q, \Sigma, \Gamma, \delta)$ and a word $\gamma \in \Gamma^*$
Decide: Whether there is a state s such that $(Q, \gamma) \Rightarrow s$

The Det-Ada-Sync problem is the same as Ada-Sync, except that the given pushdown automaton is deterministic. Notice that we can generalise the adaptive synchronising problem by the following *subset adaptive synchronising problem* (Subset-Ada-Sync): Given a PDA $\mathcal{P} = (Q, \Sigma, \Gamma, \delta)$, a subset $I \subseteq Q$ and a word $\gamma \in \Gamma^*$, decide if there is a state s such that $(I, \gamma) \Rightarrow s$. Similarly, we can define Det-Subset-Ada-Sync.

▶ Remark 1. One can also frame both of these problems in various other ways such as "Given $\mathcal{P}, \gamma$ and q does $(Q, \gamma) \Rightarrow q$?" or "Given $\mathcal{P}, \gamma, I$, is there a q such that $(I, \gamma) \overset{\text{sup}}{\Longrightarrow} q$" etc. We chose this version, because this is similar to the way it is defined for the finite-state version (Problem 1 of [17]). In order to make the lower bounds easier to understand, we introduce a few different variants of Ada-Sync and Subset-Ada-Sync in Section 3 and conclude that they are all polynomial-time equivalent with Ada-Sync. We defer a detailed analysis of the different variants of this problem to future work.

▶ Remark 2. One can relax the notion of a synchroniser and ask instead for an adaptive "homing" word, which is the same as a synchroniser, except that we now only require that if (S, γ) is the label of a leaf then S is *any* singleton. Intuitively, in an adaptive homing word, we are content with knowing the state the automaton is in after applying the strategy, rather than enforcing the automaton to synchronise into some state. To keep the discussion focused on the synchronising problem, in the main paper, we present only the results regarding Ada-Sync and Subset-Ada-Sync. In the full version of the paper, we state the homing word problem formally and prove that it is polynomial-time equivalent to Ada-Sync.

The main results of this paper are now as follows:

▶ **Theorem 3.** *Ada-Sync and Subset-Ada-Sync are both* 2-EXPTIME*-complete. Det-Ada-Sync and Det-Subset-Ada-Sync are both* EXPTIME*-complete.*

3 Equivalence of Various Formulations

In this section, we show that the problems Ada-Sync and Subset-Ada-Sync are polynomial-time equivalent to each other. A similar result is also shown for their corresponding deterministic versions. We note that such a result is not true for finite-state (Moore) machines (Table 1 of [17]) and so we provide a proof of this here, because it illustrates the significance of the stack in the pushdown version.

▶ **Lemma 4.** *Ada-Sync (resp. Det-Ada-Sync) is polynomial time equivalent to Subset-Ada-Sync (resp. Det-Subset-Ada-Sync).*

Proof. It suffices to show that Subset-Ada-Sync (resp. Det-Subset-Ada-Sync) can be reduced to Ada-Sync (resp. Det-Ada-Sync) in polynomial time.

Let $\mathcal{P} = (Q, \Sigma, \Gamma, \delta)$ be a PDA with $I \subseteq Q$ and $\gamma \in \Gamma^*$. Let q_I be some fixed state in the subset I. Construct $\mathcal{P}'$ from $\mathcal{P}$ by adding a new stack letter # and the following new transitions: Upon reading any $a \in \Sigma$, if the top of the stack is #, then any state $q \in I$ pops # and stays at q whereas any state $q \notin I$ pops # and moves to q_I. Notice that $\mathcal{P}'$ is deterministic if $\mathcal{P}$ is.

It is clear that if $(I, \gamma) \underset{\mathcal{P}}{\Longrightarrow} s$ for some state s, then $(Q, \#\gamma) \underset{\mathcal{P}'}{\Longrightarrow} s$. We now claim that the other direction is true as well. To see this, suppose there is a synchroniser in $\mathcal{P}'$ (say T) between $(Q, \#\gamma)$ and some state s. It is easy to see that, irrespective of the label of the outgoing edge from the root of T, there is only one child of the root which is labelled by (I, γ). Now, no transition pushes # onto the stack and so nowhere else in the synchroniser does # appear in the label of some vertex. It is then easy to see that if we remove the root of T, we get a synchroniser between (I, γ) and s in $\mathcal{P}$. ◀

Lemma 4 allows us to introduce a series of problems which we can prove are poly-time equivalent to Ada-Sync. The reason to consider these problems is that lower bounds for these are substantially easier to prove than for Ada-Sync. The three problems are as follows:

1. Given-Sync: Given a PDA $\mathcal{P}$, a subset I, a word γ and also *a state* s, check if $(I, \gamma) \Rightarrow s$.
2. Super-Sync has the same input as Given-Sync, except we ask if $(I, \gamma) \overset{\text{sup}}{\Longrightarrow} s$.
3. Special-Sync is the same as Super-Sync but restricted to inputs where γ is $\perp$.

▶ **Lemma 5.** *Subset-Ada-Sync, Given-Sync, Super-Sync and Special-Sync are all poly. time equivalent. Further the same applies for their corresponding deterministic versions.*

Because of this lemma, for the rest of this paper, we will only be concerned with the Special-Sync problem, where given a PDA $\mathcal{P}$, a subset I and a state s, we have to decide if $(I, \perp) \overset{\text{sup}}{\Longrightarrow} s$.

4 Lower Bounds

To prove the lower bounds, we introduce the notion of an alternating extended pushdown system (AEPS), which is an extension of pushdown systems with Boolean variables and alternation.

4.1 Alternating Extended Pushdown Systems

An *alternating extended pushdown system* (AEPS) $\mathcal{A}$ is a tuple $(Q, V, \Gamma, \Delta, \mathit{init}, \mathit{fin})$ where Q and V are finite sets of states and Boolean variables respectively, Γ is the stack alphabet, $\mathit{init}, \mathit{fin} \in Q$ are the initial and final states respectively. $\mathcal{A}$ has no input letters but it has a stack to which it can pop and push letters from Γ. Each variable in V is of Boolean type and a transition of $\mathcal{A}$ can apply simple tests on these variables and depending on the outcome, can update their values. A configuration of $\mathcal{A}$ is a tuple (q, γ, F) where $q \in Q, \gamma \in \Gamma^*$ and $F \colon V \to \{0, 1\}$ is a function assigning a Boolean value to each variable.

Let `test` denote the set of *tests* given by $\{v \overset{?}{=} b : v \in V, b \in \{0, 1\}\}$ and let `cmd` denote the set of *commands* given by $\{v \to b : v \in V, b \in \{0, 1\}\}$. A *consistent command* is a conjunction of elements from `cmd` such that for every $v \in V$, both $v \to 0$ and

$v \to 1$ are not present in `cmd`. The transition relation Δ consists of transitions of the form $(q, A, G) \hookrightarrow \{(q_1, \gamma_1, C_1), \dots, (q_k, \gamma_k, C_k)\}$ where $q, q_1, \dots, q_k \in Q$, $A \in \Gamma, \gamma_1, \dots, \gamma_k \in \Gamma^*$, G is a conjunction of elements from `test` and each C_i is a consistent command. Intuitively, at a configuration $(q, A\gamma, F)$ the machine *non-deterministically* selects a transition of the form $(q, A, G) \hookrightarrow \{(q_1, \gamma_1, C_1), \dots, (q_k, \gamma_k, C_k)\}$ such that the assignment F satisfies the conjunction G and then *forks* into k copies in the configurations $(q_1, \gamma_1\gamma, F[C_1]), \dots, (q_k, \gamma_k\gamma, F[C_k])$ where $F[C_i]$ is the function obtained by updating F according to the command C_i. With this intuition in mind, we say that a transition $(q, A, G) \hookrightarrow \{(q_1, \gamma_1, C_1), \dots, (q_k, \gamma_k, C_k)\}$ is enabled at a configuration $(p, B\gamma, F)$ iff $p = q, B = A$ and F satisfies all the tests in G.

A *run* from a configuration (q, η, H) to a configuration (q', η', H') is a tree satisfying the following properties: The root is labelled by (q, η, H). If an internal node n is labelled by the configuration $(p, A\gamma, F)$ then there exists the following transition: $(p, A, G) \hookrightarrow \{(p_1, \gamma_1, C_1), (p_2, \gamma_2, C_2), \dots, (p_k, \gamma_k, C_k)\}$ which is enabled at $(p, A\gamma, F)$ such that the children of n are labelled by $(p_1, \gamma_1\gamma, F[C_1])$,..., $(p_k, \gamma_k\gamma, F[C_k])$, where $F[C_i](v) = b$ if C_i contains a command of the form $v \to b$ and $F[C_i](v) = F(v)$ otherwise. Finally all the leaves are labelled by (q', η', H'). If a run exists between (q, η, H) and (q', η', H') then we denote it by $(q, \eta, H) \xrightarrow[\mathcal{A}]{*} (q', \eta', H')$. An *accepting run* from a configuration (q, η, H) is a run from (q, η, H) to $(\mathit{fin}, \perp, \mathbf{0})$ where $\mathbf{0}$ is the zero function. An accepting run of an AEPS is simply an accepting run from the initial configuration $(\mathit{init}, \perp, \mathbf{0})$. The emptiness problem is then to decide whether a given AEPS has an accepting run.

By a simple adaptation of the EXPTIME-hardness proof for emptiness of alternating pushdown systems which have no Boolean variables (Theorem 5.4 of [6], Prop. 31 of [23]) we prove that

▶ **Lemma 6.** *The emptiness problem for AEPS is* 2-EXPTIME*-hard.*

An AEPS $\mathcal{A}$ is called a *non-deterministic extended pushdown system* (NEPS) if every transition of $\mathcal{A}$ is of the form $(p, A, F) \hookrightarrow \{(q, \gamma, C)\}$. By Theorem 2 of [13] we have that

▶ **Lemma 7.** *The emptiness problem for NEPS is* EXPTIME*-hard.*

▶ Remark 8. The hardness result for AEPS could also be inferred from Theorem 10 of [13]. Because we use a different notation, for the sake of completeness, we provide the proofs of both of these lemmas in the full version of the paper.

4.2 Reduction from Alternating Extended Pushdown Systems

▶ **Theorem 9.** *Special-Sync, Subset-Ada-Sync and Ada-Sync are all* 2-EXPTIME*-hard. Det-Special-Sync, Det-Subset-Ada-Sync and Det-Ada-Sync are all* EXPTIME*-hard.*

In this subsection, we provide the proof sketches of Theorem 9 by a reduction from the emptiness problem for AEPS to Special-Sync. Let $\mathcal{A} = (Q, V, \Gamma, \Delta, \mathit{init}, \mathit{fin})$ be an AEPS. Without loss of generality, we can assume that if $(q, A, G) \hookrightarrow \{(q_1, \gamma_1, C_1), (q_2, \gamma_2, C_2), \dots, (q_k, \gamma_k, C_k)\} \in \Delta$, then $\gamma_i \neq \gamma_j$ for $i \neq j$. (This can be accomplished, by prefixing new characters to each γ_i, moving to some intermediate states and then popping the new characters and moving to the respective q_i's). Having made this assumption, the reduction is described below.

From the given AEPS $\mathcal{A}$, we now construct a pushdown automaton $\mathcal{P}$ as follows. The stack alphabet of $\mathcal{P}$ will be Γ. For each transition $t \in \Delta$, $\mathcal{P}$ will have an input letter `in`(t). $\mathcal{P}$ will also have another input letter `end`. The state space of $\mathcal{P}$ will be the set $Q \cup (V \times \{0, 1\}) \cup \{q_{acc}, q_{rej}\}$, where q_{acc} and q_{rej} are two states, which on reading any input letter, will leave the stack untouched and simply stay at q_{acc} and q_{rej} respectively.

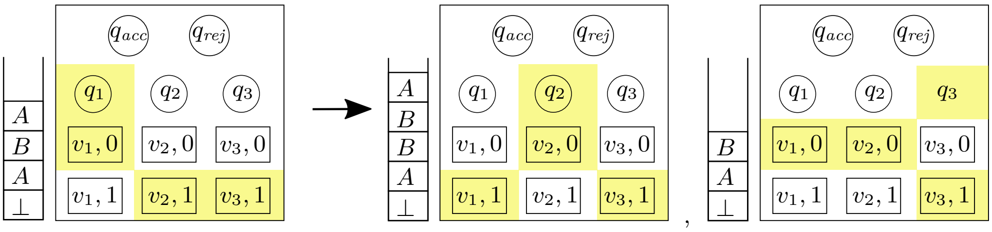


Figure 3 Let t be the transition $(q_1, A, [v_1? = 0, v_3? = 1]) \hookrightarrow \{(q_2, AB, [v_1 \leftarrow 1, v_2 \leftarrow 0]), (q_3, \epsilon, [v_2 \leftarrow 0])\}$ in $\mathcal{A}$. In $\mathcal{A}$, using t, the configuration $C_1 := (q_1, ABA\bot, [v_1 = 0, v_2 = 1, v_3 = 1])$ can fork into $C_2 := (q_2, ABBA\bot, [v_1 = 1, v_2 = 0, v_3 = 1])$ and $C_3 := (q_3, BA\bot, [v_1 = 0, v_2 = 0, v_3 = 1])$.

We now give an intuition behind the transitions of $\mathcal{P}$. Given an assignment $F : V \to \{0, 1\}$ of the Boolean variables V, and a state q of $\mathcal{A}$, we use the notation $[q, F]$ to denote the subset $\{q\} \cup \{(v, F(v)) : v \in V\}$ of states of $\mathcal{P}$. Intuitively, a configuration (q, γ, F) of $\mathcal{A}$ is simulated by its corresponding *pseudo-configuration* $([q, F], \gamma)$ in $\mathcal{P}$.

▶ **Example 10.** The caption of Figure 3 describes an example, where there is a transition t in $\mathcal{A}$, and a configuration C_1 forks into two configurations C_2 and C_3 in $\mathcal{A}$ by using t. The diagram in Figure 3 illustrates the simulation of the forking on the corresponding pseudo-configurations of C_1, C_2, C_3 that the automaton $\mathcal{P}$ will achieve when reading the letter $\mathtt{in}(t)$. The shaded part along with the stack content on the left before the arrow denotes the pseudo-configuration of C_1 and upon reading $\mathtt{in}(t)$ from this pseudo-configuration, we get two possible successors, each of which correspond to the pseudo-configurations of C_2 and C_3 respectively.

Now we give a formal description of the transitions of $\mathcal{P}$. Let $t = (q, A, G) \hookrightarrow \{(q_1, \gamma_1, C_1), \ldots, (q_k, \gamma_k, C_k)\}$ be a transition of $\mathcal{A}$. Let $p \in Q$. Upon reading $\mathtt{in}(t)$, if $p \neq q$ then p immediately moves to the q_{rej} state. Further, even state q moves to the q_{rej} state if the top of the stack is not A. However, if the top of the stack is A, then q pops A and *non-deterministically* pushes any one of $\gamma_1, \ldots, \gamma_k$ onto the stack and if it pushed γ_i, then q moves to the state q_i.

Let $(v, b) \in V \times \{0, 1\}$. Upon reading $\mathtt{in}(t)$, if the test $v \overset{?}{=} (1 - b)$ appears in the guard G, then (v, b) immediately moves to the q_{rej} state. (Notice that this is a purely syntactical condition on $\mathcal{A}$). Further, if the top of the stack is not A, then once again (v, b) moves to q_{rej}. If these two cases do not hold, then (v, b) pops A and *non-deterministically* picks an $i \in \{1, \ldots k\}$ and pushes γ_i onto the stack. Having pushed γ_i, if C_i does not update the variable v, it stays in state (v, b); otherwise if C_i has a command $v \to b'$, it moves to (v, b').

Finally, upon reading $\mathtt{end}$, the states in $[\mathit{fin}, \mathbf{0}]$ move to the q_{acc} state and all the other states in $Q \cup (V \times \{0, 1\})$ move to the q_{rej} state. Notice that there are no outgoing transitions from q_{rej} and so there is no way to move from q_{rej} to q_{acc}.

The following two facts can be easily inferred from the construction of $\mathcal{P}$:

Fact A: Suppose t is a transition of $\mathcal{A}$ which is not enabled at the configuration $(q, A\gamma, F)$. Then, upon reading $\mathtt{in}(t)$, there is at least one possible successor (S, η) of the pseudo-configuration $([q, F], A\gamma)$ such that $q_{rej} \in S$.

Fact B: Suppose the configuration $(q, A\gamma, F)$ forks into the following configurations $(q_1, \gamma_1\gamma, F_1), \ldots, (q_k, \gamma_k\gamma, F_k)$ using the transition t in the AEPS $\mathcal{A}$. Then, the possible successors from the pseudo-configuration $([q, F], A\gamma)$ upon reading $\mathtt{in}(t)$ in the PDA $\mathcal{P}$ are $([q_1, F_1], \gamma_1\gamma), \ldots, ([q_k, F_k], \gamma_k\gamma)$.

Using these 2 facts, we can then prove that $\mathcal{A}$ has an accepting run iff there is a super-synchroniser in $\mathcal{P}$ between $([init, \mathbf{0}], \bot)$ and q_{acc}. Intuitively, if we have an accepting run of $\mathcal{A}$, then the observer, using Fact B, has a strategy to force $\mathcal{P}$ into one of the states in $([fin, \mathbf{0}])$ with the stack content being $\bot$. Once she does that, she can input the letter `end` and synchronise to the state q_{acc}.

For the reverse direction, with a little case-analysis, we can show that in any super-synchroniser between $([init, \mathbf{0}], \bot)$ and q_{acc}, all non-leaf nodes must be a pseudo-configuration of some configuration in $\mathcal{A}$, and all the parents of a leaf must be labelled by $([fin, \mathbf{0}], \bot)$ Intuitively, then by Facts A and B, such a super-synchroniser must be a simulation of a run in $\mathcal{A}$ (similar to Figure 3) and hence, we can translate it back to an accepting run in $\mathcal{A}$.

Notice that $\mathcal{P}$ is deterministic if $\mathcal{A}$ is non-deterministic. Hence, by Lemmas 6 and 7, we obtain Theorem 9.

5 Upper Bounds

In this section, we will give algorithms that solve Special-Sync and Det-Special-Sync. We first give a reduction from Special-Sync to the problem of checking emptiness in an alternating pushdown system, which we define below. Then, we show that for Det-Special-Sync, the same reduction produces alternating pushdown systems with a "modular" structure, which we exploit to reduce the running time.

5.1 Adaptive Synchronisation for Non-deterministic PDA

An alternating pushdown system (APS) is an alternating extended pushdown system which has no Boolean variables. Since there are no variables, we can suppress any notation corresponding to the variables, e.g., configurations can be just denoted by (q, γ). It is known that the emptiness problem for APS is in EXPTIME (Theorem 4.1 of [4]). We now give an exponential time reduction from Special-Sync to the emptiness problem for APS.

Let $\mathcal{P} = (Q, \Sigma, \Gamma, \delta)$ be a PDA with $I \subseteq Q$, $s \in Q$. Construct the following APS $\mathcal{A}_\mathcal{P} = (2^Q, \Gamma, \Delta, I, \{s\})$ where Δ is defined as follows: Given $S \subseteq Q, a \in \Sigma$ and $A \in \Gamma$, let $E_1, \dots, E_k$ be the equivalence classes of the relation $\sim^a_{S,A}$ as defined in subsection 2.2. Then, we have the following transition in $\mathcal{A}_\mathcal{P}$:

$$(S, A) \hookrightarrow \{(next(E_1), word(E_1)), (next(E_2), word(E_2)), \dots, (next(E_k), word(E_k)\} \quad (1)$$

The following fact is immediate from the definition of a super-synchroniser and from the construction of $\mathcal{A}_\mathcal{P}$.

▶ **Proposition 11.** *Let $S \subseteq 2^Q, \gamma \in \Gamma^*$. Then a labelled tree T is a super-synchroniser between (S, γ) and s in $\mathcal{P}$ if and only if T is an accepting run from (S, γ) in $\mathcal{A}_\mathcal{P}$.*

By Theorem 4.1 of [4], emptiness for APS can be solved in exponential time and so

▶ **Theorem 12.** *Special-Sync is in* 2-EXPTIME

5.2 Adaptive Synchronisation for Deterministic PDA

Let $\mathcal{P} = (Q, \Sigma, \Gamma, \delta)$ be a deterministic PDA with $I \subseteq Q, s \in Q$. We have the following proposition, whose proof follows from the fact that $\mathcal{P}$ is deterministic.

▶ **Proposition 13.** *Suppose $S \subseteq Q, a \in \Sigma, A \in \Gamma$ and suppose $E_1, \dots, E_k$ are the equivalence classes of $\sim^a_{S,A}$. Then, $|S| \geq \sum_{i=1}^{k} |next(E_i)|$.*

Now, given $\mathcal{P}$, consider the APS $\mathcal{A}_{\mathcal{P}} = (2^Q, \Gamma, \Delta, I, \{s\})$ that we have constructed in subsection 5.1. By Proposition 13, we now have the following lemma.

▶ **Lemma 14.** *For any $S \in 2^Q, \gamma \in \Gamma^*$, any accepting run of $\mathcal{A}_{\mathcal{P}}$ from the configuration (S, γ) has at most $|S|$ leaves.*

The following corollary follows from the lemma above.

▶ **Corollary 15.** *Any accepting run of $\mathcal{A}_{\mathcal{P}}$ has at most $|I|$ leaves.*

▶ **Example 16.** Let $\mathcal{P}$ be the deterministic PDA from Figure 1. Figure 4 shows an example of an accepting run in the corresponding APS $\mathcal{A}_{\mathcal{P}}$ from $I := \{1, 2, 3, 4\}$. Notice that there are $|I| = 4$ leaves in this run.

Corollary 15 motivates the study of the following problem, which we call the *sparse emptiness* problem for APSs (Sparse-Empty):

Given: An APS $\mathcal{A}$ and a number k in unary.
Decide: Whether there exists an accepting run for $\mathcal{A}$ with at most k leaves

We prove the following theorem about Sparse-Empty in the next section.

▶ **Theorem 17.** *Given $\mathcal{A}$ and k, the Sparse-Empty problem can be solved in time $O(|\mathcal{A}|^{ck^2})$ for a fixed constant c.*

Now, because of Proposition 13 and because of the structure of the transitions of $\mathcal{A}_{\mathcal{P}}$ (as given by equation (1)), it is sufficient to restrict the construction of $\mathcal{A}_{\mathcal{P}}$ to only those states which have cardinality at most $|I|$ and hence, it can be assumed that $|\mathcal{A}_{\mathcal{P}}| \leq |\mathcal{P}|^{4|I|}$. This fact, along with Proposition 11, Corollary 15 and Theorem 17 implies the following theorem.

▶ **Theorem 18.** *Given an instance $(\mathcal{P}, I, s)$ of Det-Special-Sync, checking if $(I, \perp) \xRightarrow[\mathcal{P}]{sup} s$ in time $O(n^{4ck^3})$ where $n = |\mathcal{P}|$ and $k = |I|$ and c is some fixed constant.*

▶ Remark 19. Note that the algorithm to solve Det-Special-Sync on an instance $(\mathcal{P}, I, s)$, although in EXPTIME, is polynomial if $|I|$ is fixed and quasi-polynomial if $|I|$ is $O(\log |\mathcal{P}|)$.

5.3 "Sparse Emptiness" Checking of Alternating Systems

This subsection is dedicated to proving Theorem 17. We fix an alternating pushdown system $\mathcal{A} = (Q, \Gamma, \Delta, \mathit{init}, \mathit{fin})$ and a number k for the rest of this subsection. A k-accepting run of $\mathcal{A}$ is defined to be an accepting run of $\mathcal{A}$ with at most k leaves. We now split the desired algorithm for Sparse-Empty into three parts. Finally, we give its runtime analysis.

Compressing k-accepting runs of $\mathcal{A}$

We define a non-deterministic pushdown system (NPS) to be a non-deterministic extended pushdown system which has no Boolean variables. From $\mathcal{A}$, we can derive a NPS obtained by deleting all transitions which produces a universal branching, i.e, of the form $(q, A) \hookrightarrow \{(q_1, \gamma_1), \ldots, (q_k, \gamma_k)\}$ with $k > 1$. We will denote this NPS by $\mathcal{N}$. Emptiness of NPS is known to be solvable in polynomial time (Theorem 2.1 of [4]). To exploit this fact for our problem, we propose the following notion of a *compressed* accepting run of $\mathcal{A}$. Intuitively, a compressed accepting run is obtained from an accepting run of $\mathcal{A}$ by "compressing" a series of transitions belonging to the non-deterministic part $\mathcal{N}$, into a single transition. An intuition of a compressed accepting run is captured by Figure 5, which is obtained by compressing the run depicted in Figure 4.

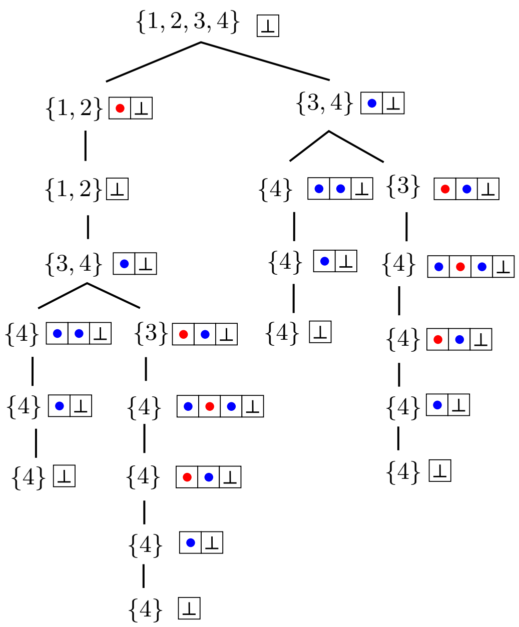


Figure 4 An accepting run of $\mathcal{A}_\mathcal{P}$ for the deterministic PDA $\mathcal{P}$ given in Figure 1.

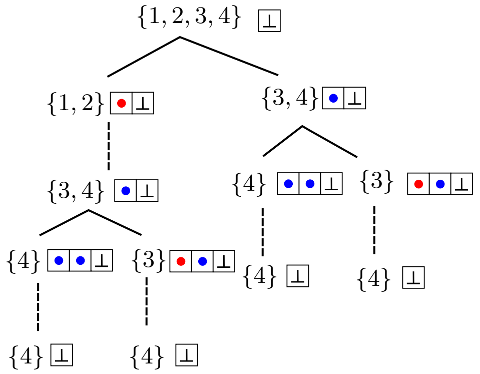


Figure 5 A compressed accepting run of $\mathcal{A}_\mathcal{P}$ for the deterministic PDA $\mathcal{P}$ given in Figure 1, obtained by compressing the run from Figure 4.

Given a tree, we say that a vertex v in the tree is *simple* if it has exactly one child and otherwise we say that it is *complex* (Note that all leaves are complex). A *compressed accepting run* of $\mathcal{A}$ from the configuration (p, η) is a labelled tree such that: The root is labelled by (p, η). If v is a simple vertex labelled by (q, γ) and u is its only child labelled by (q', γ') then u is a complex vertex and $(q, \gamma) \xrightarrow[\mathcal{N}]{*} (q', \gamma')$. If v is a complex vertex labelled by $(q, A\gamma)$ and $v_1, \dots, v_k$ are its children with $k > 1$, then there is a transition $(q, A) \hookrightarrow \{(q_1, A_1), \dots, (q_k, A_k)\}$ in $\mathcal{A}$ such that the label of v_i is $(q_i, A_i\gamma)$. Finally, all the leaves are labelled by $(\mathit{fin}, \bot)$. A compressed accepting run of $\mathcal{A}$ is a compressed accepting run from $(\mathit{init}, \bot)$ and a k-compressed accepting run is a compressed accepting run with at most k leaves. We now have the following lemma.

▶ **Lemma 20.** *There is a k-accepting run of $\mathcal{A}$ from a configuration (p, η) iff there is a k-compressed accepting run of $\mathcal{A}$ from (p, η).*

Searching for k-compressed accepting runs

To fully use the result of Lemma 20, we need some results about non-deterministic pushdown systems, which we state here. Recall that $\mathcal{N}$ is an NPS over the states Q and stack alphabet Γ obtained from the APS $\mathcal{A}$. We say that $M = (Q^M, \Gamma, \delta^M, F^M)$ is an $\mathcal{N}$-*automaton* if M is a non-det. finite-state automaton over the alphabet Γ with accepting states F^M such that for each state $q \in Q$, there is a unique state $q^M \in Q^M$. The set of configurations of $\mathcal{A}$ that are stored by M (denoted by $\mathcal{C}(M)$) is defined to be the set $\{(q, \gamma) : \gamma$ is accepted in M from the state $q^M\}$. In the above definition, note that Q^M can potentially have more states other than the set $\{q^M \mid q \in Q\}$.

▶ **Example 21.** Let us consider the pushdown automaton in Figure 1, and let $\mathcal{N}$ be the NPS obtained by ignoring the input alphabets $\square$ and $\diamond$. Then observe that from all the states 1,2,3 and 4, with any content on the stack, one can reach state 4 with an empty stack, by popping out all the elements. So, the set of configurations from wich there is an accepting run is $\{(i, \gamma) \mid i \in \{1, 2, 3, 4\}, \gamma \in \bot \cdot \{\bullet, \bullet\}^*\}$. One can define the $\mathcal{N}$-automaton M for it, as an automaton with five states $\{q_1, q_2, q_3, q_4, q_f\}$ where q_f is a final state and each of q_1, q_2, q_3 and q_4 on reading $\bot$ goes to q_f and stays in q_f on reading $\bullet$ or $\bullet$. It is easy to see that this automaton accepts all words of the form $\bot \cdot \{\bullet, \bullet\}^*$.

▶ **Theorem 22** (Section 2.3 and Theorem 2.1 of [4])**.** *Given an $\mathcal{N}$-automaton M, in time polynomial in $\mathcal{N}$ and M, we can construct an $\mathcal{N}$-automaton M' which has the* same states *as M such that M' stores the set of predecessors of M, i.e., $\mathcal{C}(M') = \{(q', \gamma') : \exists (q, \gamma) \in \mathcal{C}(M)$ such that $(q', \gamma') \xrightarrow[\mathcal{N}]{*} (q, \gamma)\}$.*

We say that an unlabelled tree is *structured*, if the child of every simple vertex is a complex vertex. An ℓ-structured tree is simply a structured tree which has at most ℓ leaves. Notice that the height of an ℓ-structured tree is $O(\ell)$ and since it has at most ℓ leaves, it follows that a ℓ-structured tree can be described using a polynomial number of bits in ℓ. Hence, the number of ℓ-structured trees is $O(2^{\ell^c})$ for some fixed c.

Now let us come back to the problem of searching for k-accepting runs of $\mathcal{A}$. By Lemma 20 it suffices to search for a k-compressed accepting run of $\mathcal{A}$. Notice that if we take a k-compressed accepting run and remove its labels, we get a k-structured tree. Now, suppose we have an algorithm `Check` that takes a k-structured tree T and checks if T can be labelled to make it a k-compressed accepting run of $\mathcal{A}$. Then, by calling `Check` on every k-structured tree, we have an algorithm to check for the existence of a k-compressed accepting run of $\mathcal{A}$. Hence, it suffices to describe this procedure `Check` which is what we will do now.

The algorithm `Check`

Let T be a k-structured tree. For each vertex v in the tree T, `Check` will assign a $\mathcal{N}$-automaton M_v such that M_v will have the following property:

> Invariant (*) : A configuration $(q, \gamma) \in \mathcal{C}(M_v)$ iff all the vertices of the subtree rooted at v can be labelled such that the resulting labelled subtree is a compressed accepting run of $\mathcal{A}$ from (q, γ).

The construction of each M_v is as follows: Let Q be the states and Δ be the transitions of the alternating pushdown system $\mathcal{A}$.

- Suppose vertex v is a leaf. We let M_v be an automaton such that $\mathcal{C}(M_v) = \{(\mathit{fin}, \bot)\}$. Notice that such a M_v can be easily constructed in polynomial time.
- Suppose vertex v is simple and u is its child. We take M_u and use Theorem 22 to construct the $\mathcal{N}$-automaton M_v. Note that M_v has the same set of states as M_u.
- Suppose v is complex and suppose $v_1, \ldots, v_\ell$ are its children. For each $1 \le i \le \ell$ and for every configuration (q, γ) of $\mathcal{A}$, let $\delta_i(q^{M_{v_i}}, \gamma)$ denote the set of states that the automaton M_{v_i} will be in after reading γ from the state $q^{M_{v_i}}$. To construct M_v first do a product construction $M_{v_1} \times M_{v_2} \times \cdots \times M_{v_\ell}$, so that the resulting product automaton stores precisely the set of configurations which are stored by each of the individual automata $M_{v_1}, \ldots, M_{v_\ell}$. Then, for each $q \in Q$, add a state q^{M_v}. Then for each transition $(p, A) \hookrightarrow \{(p_1, \gamma_1), \ldots, (p_\ell, \gamma_\ell)\}$ in Δ, add a transition in M_v, which upon reading A, takes p^{M_v} to any of the states in $\delta_1({p_1}^{M_{v_1}}, \gamma_1) \times \delta_2({p_2}^{M_{v_2}}, \gamma_2) \times \cdots \times \delta_l({p_\ell}^{M_{v_\ell}}, \gamma_\ell)$. Intuitively, we accept a word $A\gamma$ from the state p^{M_v} if for each i, the word $\gamma_i\gamma$ can be accepted from the state ${p_i}^{M_{v_i}}$.

▶ **Proposition 23.** *For each vertex v of the tree T, M_v satisfies invariant (*)*

Finally, we accept iff $(init, \bot) \in \mathcal{C}(M_r)$ where r is the root of the tree. The correctness of `Check` follows from the proposition above.

Running time analysis

Let us analyse the running time of `Check`. Let T be a k-structured tree and therefore T has $O(k^2)$ vertices. `Check` assigns to each vertex v of T an automaton M_v. We claim that the running time of `Check` is $O(k^2 \cdot |\mathcal{A}|^{ck^2})$ (for some fixed constant c) because of the following facts:

1) By induction on the structure of the tree T, it can be proved that, there exists a constant d, such that if h_v is the height of a vertex v and l_v is the number of leaves in the sub-tree of v, then the number of states of M_v is $O(|\mathcal{A}|^{dh_v l_v})$ (Recall that $h_v l_v$ is at most $O(k^2)$).

2) If an $\mathcal{N}$-automaton has n states, then the number of transitions it can have is $O(|\mathcal{A}| \cdot n^2)$.

3) For a vertex v with children $v_1, \ldots, v_\ell$, M_v can be constructed in polynomial time in the size of $|M_{v_1}| \times |M_{v_2}| \times \ldots |M_{v_\ell}|$ and $|\mathcal{A}|$.

Notice that everything else apart from Fact 1) is easy to see. To prove Fact 1), we proceed by bottom-up induction on the structure of the tree T. For the base case when the vertex v is a leaf, notice that we can easily construct the required automaton M_v with at most $O(|\mathcal{A}|)$ states. Suppose, v is a simple vertex and u its only child. By Theorem 22, M_v has the same set of states as M_u. By induction hypothesis, the number of states of M_u is $O\left(|\mathcal{A}|^{dh_u l_u}\right)$ and so the number of states of M_v is $O\left(|\mathcal{A}|^{dh_v l_v}\right)$. Suppose v is a complex vertex and $v_1, \ldots, v_\ell$ are its children. Let h be the maximum height amongst the vertices $v_1, \ldots, v_\ell$. By induction hypothesis, the number of states of each M_{v_i} is $O\left(|\mathcal{A}|^{dhl_{v_i}}\right)$. It is then clear that the number of states of M_v is $O\left(\prod_{i=1}^{\ell} |\mathcal{A}|^{dhl_{v_i}} + |\mathcal{A}|\right) = O\left(|\mathcal{A}|^{dhl_v} + |\mathcal{A}|\right) = O\left(|\mathcal{A}|^{d(h+1)l_v}\right) = O\left(|\mathcal{A}|^{dh_v l_v}\right)$.

Now the final algorithm for Sparse-Empty simply iterates over all k-structured trees and calls `Check` on all of them. Since the number of k-structured trees is at most $f(k)$ where f is an exponential function, it follows that the total running time is $O\left(f(k) \cdot k^2 \cdot |\mathcal{A}|^{ck^2}\right) = O(|\mathcal{A}|^{ek^2})$ for some constant e.

6 Conclusion

Our results can be considered as a step in the research direction recently proposed by Fernau, Wolf and Yamakami in [12], in which the authors prove that the synchronisation problem for PDAs is undecidable when the stack is *not* visible. They also suggest looking into different variants of synchronisation for PDAs with a view towards the decidability and complexity frontier. Within this context, we believe we have proposed a natural variant of synchronisation in which the observer can see the stack and given decidability and complexity-theoretic optimal results for both the non-deterministic and the deterministic cases.

As future work, it might be interesting to consider the adaptive synchronising problem for subclasses of pushdown automata such as one-counter automata and visibly pushdown automata. It might also be interesting to consider the problem of looking for *short* adaptive synchronisers, i.e., adaptive synchronisers whose size is not bigger than a given bound.

References

1 A. R. Balasubramanian and K. S. Thejaswini. Adaptive synchronisation of pushdown automata, 2021. arXiv:2102.06897.

2 Marie-Pierre Béal, Eugen Czeizler, Jarkko Kari, and Dominique Perrin. Unambiguous automata. *Math. Comput. Sci.*, 1(4):625–638, 2008. doi:10.1007/s11786-007-0027-1.

3 Yaakov Benenson, Rivka Adar, Tamar Paz-Elizur, Zvi Livneh, and Ehud Shapiro. Dna molecule provides a computing machine with both data and fuel. *Proceedings of the National Academy of Sciences*, 100(5):2191–2196, 2003.

4 Ahmed Bouajjani, Javier Esparza, and Oded Maler. Reachability analysis of pushdown automata: Application to model-checking. In Antoni W. Mazurkiewicz and Józef Winkowski, editors, *CONCUR '97: Concurrency Theory, 8th International Conference, Warsaw, Poland, July 1-4, 1997, Proceedings*, volume 1243 of *Lecture Notes in Computer Science*, pages 135–150. Springer, 1997. doi:10.1007/3-540-63141-0_10.

5 Manfred Broy, Bengt Jonsson, Joost-Pieter Katoen, Martin Leucker, and Alexander Pretschner, editors. *Model-Based Testing of Reactive Systems, Advanced Lectures [The volume is the outcome of a research seminar that was held in Schloss Dagstuhl in January 2004]*, volume 3472 of *LNCS*. Springer, 2005.

6 Ashok K. Chandra, Dexter Kozen, and Larry J. Stockmeyer. Alternation. *J. ACM*, 28(1):114–133, 1981. doi:10.1145/322234.322243.

7 Dmitry Chistikov, Pavel Martyugin, and Mahsa Shirmohammadi. Synchronizing automata over nested words. *J. Autom. Lang. Comb.*, 24(2-4):219–251, 2019. doi:10.25596/jalc-2019-219.

8 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, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 121–132. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.121.

9 Laurent Doyen, Thierry Massart, and Mahsa Shirmohammadi. The complexity of synchronizing markov decision processes. *J. Comput. Syst. Sci.*, 100:96–129, 2019. doi:10.1016/j.jcss.2018.09.004.

10 David Eppstein. Reset sequences for monotonic automata. *SIAM J. Comput.*, 19(3):500–510, 1990.

11 Henning Fernau and Petra Wolf. Synchronization of deterministic visibly push-down automata. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2020, December 14-18, 2020, BITS Pilani, K K Birla Goa Campus, Goa, India (Virtual Conference)*, volume 182 of *LIPIcs*, pages 45:1–45:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.FSTTCS.2020.45.

12 Henning Fernau, Petra Wolf, and Tomoyuki Yamakami. Synchronizing deterministic push-down automata can be really hard. In Javier Esparza and Daniel Král', editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 33:1–33:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.33.

13 Patrice Godefroid and Mihalis Yannakakis. Analysis of boolean programs. In Nir Piterman and Scott A. Smolka, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7795 of *Lecture Notes in Computer Science*, pages 214–229. Springer, 2013. doi:10.1007/978-3-642-36742-7_16.

14 F. C. Hennine. Fault detecting experiments for sequential circuits. In *1964 Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 95–110, 1964.

15 Chris Keeler and Kai Salomaa. Alternating finite automata with limited universal branching. In Alberto Leporati, Carlos Martín-Vide, Dana Shapira, and Claudio Zandron, editors, *Language and Automata Theory and Applications - 14th International Conference, LATA 2020, Milan, Italy, March 4-6, 2020, Proceedings*, volume 12038 of *Lecture Notes in Computer Science*, pages 196–207. Springer, 2020. doi:10.1007/978-3-030-40608-0_13.

16 Arun Lakhotia, Eric Uday Kumar, and M. Venable. A method for detecting obfuscated calls in malicious binaries. *IEEE Transactions on Software Engineering*, 31(11):955–968, 2005. doi:10.1109/TSE.2005.120.

17 Kim Guldstrand Larsen, Simon Laursen, and Jirí Srba. Synchronizing strategies under partial observability. In 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*, pages 188–202. Springer, 2014. doi:10.1007/978-3-662-44584-6_14.

18 D. Lee and M. Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

19 Balas K Natarajan. An algorithmic approach to the automated design of parts orienters. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 132–142. IEEE, 1986.

20 Karin Quaas and Mahsa Shirmohammadi. Synchronizing data words for register automata. *ACM Trans. Comput. Log.*, 20(2):11:1–11:27, 2019. doi:10.1145/3309760.

21 Fu Song and Tayssir Touili. Pushdown model checking for malware detection. *Int. J. Softw. Tools Technol. Transf.*, 16(2):147–173, 2014. doi:10.1007/s10009-013-0290-1.

22 Mikhail V. Volkov. Synchronizing automata and the cerny conjecture. In Carlos Martín-Vide, Friedrich Otto, and Henning Fernau, editors, *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers*, volume 5196 of *Lecture Notes in Computer Science*, pages 11–27. Springer, 2008. doi:10.1007/978-3-540-88282-4_4.

23 Igor Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001. doi:10.1006/inco.2000.2894.

24 Ján Černý. Poznámka k homogénnym experimentom s konečnými automatmi. *Matematicko-fyzikálny časopis*, 14(3):208–216, 1964.

Decomposing Permutation Automata

Ismaël Jecker
Institute of Science and Technology, Klosterneuburg, Austria

Nicolas Mazzocchi
IMDEA Software Institute, Madrid, Spain

Petra Wolf
Fachbereich IV, Informatikwissenschaften, Universität Trier, Germany

Abstract

A deterministic finite automaton (DFA) $\mathcal{A}$ is composite if its language $L(\mathcal{A})$ can be decomposed into an intersection $\bigcap_{i=1}^{k} L(\mathcal{A}_i)$ of languages of smaller DFAs. Otherwise, $\mathcal{A}$ is prime. This notion of primality was introduced by Kupferman and Mosheiff in 2013, and while they proved that we can decide whether a DFA is composite, the precise complexity of this problem is still open, with a doubly-exponential gap between the upper and lower bounds. In this work, we focus on permutation DFAs, i.e., those for which the transition monoid is a group. We provide an NP algorithm to decide whether a permutation DFA is composite, and show that the difficulty of this problem comes from the number of non-accepting states of the instance: we give a fixed-parameter tractable algorithm with the number of rejecting states as the parameter. Moreover, we investigate the class of commutative permutation DFAs. Their structural properties allow us to decide compositionality in NL, and even in LOGSPACE if the alphabet size is fixed. Despite this low complexity, we show that complex behaviors still arise in this class: we provide a family of composite DFAs each requiring polynomially many factors with respect to its size. We also consider the variant of the problem that asks whether a DFA is *k-factor composite*, that is, decomposable into k smaller DFAs, for some given integer $k \in \mathbb{N}$. We show that, for commutative permutation DFAs, restricting the number of factors makes the decision computationally harder, and yields a problem with tight bounds: it is NP-complete. Finally, we show that in general, this problem is in PSPACE, and it is in LOGSPACE for DFAs with a singleton alphabet.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Problems, reductions and completeness

Keywords and phrases Deterministic finite automata (DFA), Permutation automata, Commutative languages, Decomposition, Regular Languages, Primality

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.18

Related Version A version of this paper with all proofs is available at `https://arxiv.org/abs/2107.04683`

Funding *Ismaël Jecker*: Marie Skłodowska-Curie Grant Agreement No. 754411.
Nicolas Mazzocchi: BOSCO project PGC2018-102210-B-I00 (MCIU/AEI/FEDER, UE), BLOQUES-CM project S2018/TCS-4339, and MINECO grant RYC-2016-20281.
Petra Wolf: DFG project FE 560/9-1.

1 Introduction

Compositionality is a fundamental notion in numerous fields of computer science [3]. This principle can be summarised as follows: Every system should be designed by composing simple parts such that the meaning of the system can be deduced from the meaning of its parts, and how they are combined. For instance, this is a crucial aspect of modern software engineering: a program split into simple modules will be quicker to compile and easier to maintain. The use of compositionality is also essential in theoretical computer

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 18; pp. 18:1–18:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

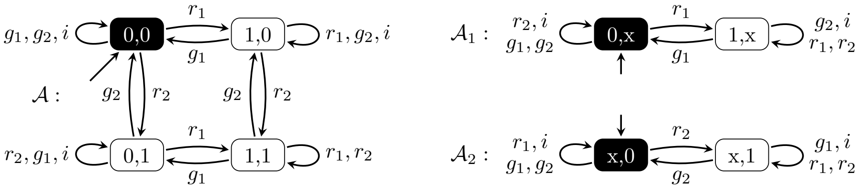


Figure 1 DFAs recognising specifications. Accepting states are drawn in black. The DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ check that every request of the first, resp. second, client is eventually granted, $\mathcal{A}$ checks both.

science: it is used to avoid the *state explosion* issues that usually happen when combining parallel processes together, and also to overcome the *scalability* issues of problems with a high theoretical complexity. In this work, we study compositionality in the setting of formal languages: we show how to make languages simpler by decomposing them into *intersections* of smaller languages. This is motivated by the *model-checking* problems. For instance, the LTL model-checking problem asks, given a linear temporal logic formula φ and a finite state machine M, whether every execution of M satisfies φ. This problem is decidable, but has a high theoretical complexity (PSPACE) with respect to the size of φ [1]. If φ is too long, it cannot be checked efficiently. This is where compositionality comes into play: if we can decompose the specification language into an intersection of simple languages, that is, decompose φ into a conjunction $\varphi = \varphi_1 \wedge \varphi_2 \wedge \cdots \wedge \varphi_k$ of small specifications, it is sufficient to check whether all the φ_i are satisfied separately.

Our aim is to develop the theoretical foundations of the compositionality principle for formal languages by investigating how to decompose into simpler parts one of the most basic model of abstract machines: deterministic finite automata (DFAs). We say that a DFA $\mathcal{A}$ is *composite* if its language can be decomposed into the intersection of the languages of smaller DFAs. More precisely, we say that $\mathcal{A}$ is *k-factor composite* if there exist k DFAs $(\mathcal{A}_i)_{1 \leq i \leq k}$ with less states than $\mathcal{A}$ such that $L(\mathcal{A}) = \bigcap_{i=1}^{k} L(\mathcal{A}_i)$. We study the two following problems:

DFA Decomp
Given: DFA $\mathcal{A}$.
Question: Is $\mathcal{A}$ composite?

DFA Bound-Decomp
Given: DFA $\mathcal{A}$ and integer $k \in \mathbb{N}$.
Question: Is $\mathcal{A}$ k-factor composite?

The next example shows that decomposing DFAs can result in substantially smaller machines.

Example. Consider Figure 1. We simulate the interactions between a system and two clients by using finite words on the alphabet $\{r_1, r_2, g_1, g_2, i\}$: At each time step, the system either receives a request from a client (r_1, r_2), grants the open requests of a client (g_1, g_2), or stays idle (i). A basic property usually required is that every request is eventually granted. This specification is recognised by the DFA $\mathcal{A}$, which keeps track in its state of the current open requests, and only accepts if none is open when the input ends. Alternatively, this specification can be decomposed into the intersection of the languages defined by the DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$: each one checks that the requests of the corresponding client are eventually granted. While in this precise example both ways of defining the specification are comparable, the latter scales drastically better than the former when the number of clients increases: Suppose that there are now $n \in \mathbb{N}$ clients. In order to check that all the requests are granted with a single DFA, we need 2^n states to keep track of all possible combinations of open requests, which is impractical when n gets too big. However, decomposing this specification into an intersection yields n DFAs of size two, one for each client. Note that, while in this specific example the decomposition is obvious, in general computing such a conjunctive form can be challenging: currently the best known algorithm needs exponential space.

	Decomp	Bound-Decomp
DFAs	EXPSPACE [9]	**PSPACE**
Permutation DFAs	**NP/FPT**	**PSPACE**
Commutative permutation DFAs	**NL**	**NP-complete**
Unary DFAs	LOGSPACE [7]	**LOGSPACE**

Figure 2 Complexity of studied problems with containing classes, with our contribution in **bold**.

DFAs in hardware. Our considered problems are of great interest in hardware implementations of finite state machines [13] where realizing large DFAs poses a challenge [5]. In [2] the authors describe a state machine language for describing complex finite state hardware controllers, where the compiled state tables can automatically be input into a temporal logic model checker. If the control mechanism of the initial finite state machine can be split up into a conjunction of constraints, considering a decomposition instead could improve this work-flow substantially. Decomposing a complex DFA $\mathcal{A}$ can lead to a smaller representation of the DFA in total, as demonstrated in the previous example in Figure 1, and on top of that the individual smaller DFAs $\mathcal{A}_i$ in the decomposition $L(\mathcal{A}) = \bigcap_{i=1}^{k} L(\mathcal{A}_i)$ can be placed independently on a circuit board, as they do not have to interact with each other and only need to read their common input from a global bus and signal acceptance as a flag to the bus. This allows for a great flexibility in circuit designs, as huge DFAs can be broken down into smaller blocks which fit into niches giving space for inflexible modules such as CPU cores.

Reversible DFAs. We focus our study on *permutation* DFAs, which are DFAs whose transition monoids are groups: each letter induces a one-to-one map from the state set into itself. These DFAs are also called *reversible* DFAs [8, 14]. Reversibility is stronger than determinism: this powerful property allows to deterministically navigate *back and forth* between the steps of a computation. This is particularly relevant in the study of the physics of computation, since irreversibility causes energy dissipation [10]. Remark that in the setting of DFAs, this power results in a loss of expressiveness: contrary to more powerful models (for instance Turing machines), reversible DFAs are less expressive than general DFAs.

Related work. The DFA Decomp problem was first introduced in 2013 by Kupferman and Moscheiff [9]. They proved that it is decidable in EXPSPACE, but left open the exact complexity: the best known lower bound is hardness for NL. They gave more efficient algorithms for restricted domains: a PSPACE algorithm for *permutation* DFAs, and a PTIME algorithm for *normal* permutation DFAs, a class of DFAs that contains all *commutative* permutation DFAs. Recently, the Decomp problem was proved to be decidable in LOGSPACE for DFAs with a singleton alphabet [7]. The trade-off between number and size of factors was studied in [12], where automata showing extreme behavior are presented, i.e., DFAs that can either be decomposed into a large number of small factors, or a small number of large factors.

Contribution. We expand the domain of instances over which the Decomp problem is tractable. We focus on permutation DFAs, and we propose new techniques that improve the known complexities. All proofs omitted due to space restrictions can be found in the full version. Unless specified otherwise, the complexity of our algorithms do not depend on the size of the alphabet of the DFA. Our results, summarised by Figure 2, are presented as follows.

Section 3. We give an NP algorithm for permutation DFAs, and we show that the complexity is directly linked to the number of non-accepting states. This allows us to obtain a fixed-parameter tractable algorithm with respect to the number of non-accepting states (Theorem 1). Moreover, we prove that permutation DFAs with a prime number of states cannot be decomposed (Theorem 2).

Section 4. We consider *commutative* permutation DFAs, where the DECOMP problem was already known to be tractable, and we lower the complexity from PTIME to NL, and even LOGSPACE if the size of the alphabet is fixed (Theorem 9). While it is easy to decide whether a commutative permutation DFA is composite, we show that rich and complex behaviours still appear in this class: there exist families of composite DFAs that require polynomially many factors to get a decomposition. More precisely, we construct a family $(\mathcal{A}_n^m)_{m,n\in\mathbb{N}}$ of composite DFAs such that $\mathcal{A}_n^m$ is a DFA of size n^m that is $(n-1)^{m-1}$-factor composite but not $(n-1)^{m-1} - 1$-factor composite (Theorem 10). Note that, prior to this result, only families of composite DFAs with sublogarithmic width were known [7].

Section 5. Finally, we study the BOUND-DECOMP problem. High widths are undesirable for practical purposes: dealing with a huge number of small DFAs might end up being more complex than dealing with a single DFA of moderate size. The BOUND-DECOMP problem copes with this issue by limiting the number of factors allowed in the decompositions. We show that this flexibility comes at a cost: somewhat surprisingly, this problem is NP-complete for commutative permutation DFAs (Theorem 17), a setting where the DECOMP problem is easy. We also show that this problem is in PSPACE for the general setting (Theorem 16), and in LOGSPACE for unary DFAs i.e. with a singleton alphabet (Theorem 18).

2 Definitions

We denote by $\mathbb{N}$ the set of non-negative integers $\{0, 1, 2, \ldots\}$. For a word $w = w_1 w_2 \ldots w_n$ with $w_i \in \Sigma$ for $1 \leq i \leq n$, we denote with $w^R = w_n \ldots w_2 w_1$ the *reverse* of w. Moreover, for every $\sigma \in \Sigma$, we denote by $\#_\sigma(w)$ the number of times the letter σ appears in w. A natural number $n > 1$ is called *composite* if it is the product of two smaller numbers, otherwise we say that n is *prime*. Two integers $m, n \in \mathbb{N}$ are called *co-prime* if their greatest common divisor is 1. We will use the following well known results [6, 11]:

Bertrand's Postulate. For all $n > 3$ there is a prime number p satisfying $n < p < 2n - 2$.

Bézout's Identity. For every pair of integers $m, n \in \mathbb{N}$, the set $\{\lambda m - \mu n \mid \lambda, \mu \in \mathbb{N}\}$ contains exactly the multiples of the greatest common divisor of m and n.

Deterministic finite automata. A *deterministic finite automaton* (DFA hereafter) is a 5-tuple $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$, where Q is a finite set of states, Σ is a finite non-empty alphabet, $\delta\colon Q \times \Sigma \to Q$ is a transition function, $q_I \in Q$ is the initial state, and $F \subseteq Q$ is a set of accepting states. The states in $Q \setminus F$ are called *rejecting* states. We extend δ to words in the expected way, thus $\delta\colon Q \times \Sigma^* \to Q$ is defined recursively by $\delta(q, \varepsilon) = q$ and $\delta(q, w_1 w_2 \cdots w_n) = \delta(\delta(q, w_1 w_2 \cdots w_{n-1}), w_n)$. The *run* of $\mathcal{A}$ on a word $w = w_1 \ldots w_n$ is the sequence of states $s_0, s_1, \ldots, s_n$ such that $s_0 = q_I$ and for each $1 \leq i \leq n$ it holds that $\delta(s_{i-1}, w_i) = s_i$. Note that $s_n = \delta(q_I, w)$. The DFA $\mathcal{A}$ *accepts* w iff $\delta(q_I, w) \in F$. Otherwise,

$\mathcal{A}$ *rejects* w. The set of words accepted by $\mathcal{A}$ is denoted $L(\mathcal{A})$ and is called the *language of* $\mathcal{A}$. A language accepted by some DFA is called a *regular language*.

We refer to the size of a DFA $\mathcal{A}$, denoted $|\mathcal{A}|$, as the number of states in $\mathcal{A}$. A DFA $\mathcal{A}$ is *minimal* if every DFA $\mathcal{B}$ such that $L(\mathcal{B}) = L(\mathcal{A})$ satisfies $|\mathcal{B}| \geq |\mathcal{A}|$.

Composite DFAs. We call a DFA $\mathcal{A}$ *composite* if there exists a family $(\mathcal{B}_i)_{1 \leq i \leq k}$ of DFAs with $|\mathcal{B}_i| < |\mathcal{A}|$ for all $1 \leq i \leq k$ such that $L(\mathcal{A}) = \bigcap_{1 \leq i \leq k} L(\mathcal{B}_i)$ and call the family $(\mathcal{B}_i)_{1 \leq i \leq k}$ a *decomposition* of $\mathcal{A}$. Note that, all $\mathcal{B}_i$ in the decomposition satisfy $|\mathcal{B}_i| < |\mathcal{A}|$ and $L(\mathcal{A}) \subseteq L(\mathcal{B}_i)$. Such DFAs are called *factors* of $\mathcal{A}$, and $(\mathcal{B}_i)_{1 \leq i \leq k}$ is also called a k*-factor decomposition* of $\mathcal{A}$. The *width* of $\mathcal{A}$ is the smallest k for which there is a k-factor decomposition of $\mathcal{A}$, and we say that $\mathcal{A}$ is k*-factor composite* iff $\mathit{width}(\mathcal{A}) \leq k$. We call a DFA $\mathcal{A}$ *prime* if it is not composite. We call a DFA $\mathcal{A}$ *trim* if all of its states are accessible from the initial state. As every non-trim DFA $\mathcal{A}$ is composite, we assume all given DFAs to be trim in the following.

We call a DFA a *permutation DFA* if for each letter $\sigma \in \Sigma$, the function mapping each state q to the state $\delta(q, \sigma)$ is a bijection. For permutation DFAs the transition monoid is a group. Further, we call a DFA $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ a *commutative DFA* if $\delta(q, uv) = \delta(q, vu)$ for every state q and every pair of words $u, v \in \Sigma^*$. In the next sections we discuss the problem of being composite for the classes of permutation DFA, and commutative permutation DFAs.

3 Decompositions of Permutation DFAs

In this section, we study permutation DFAs. Our main contribution is an algorithm for the Decomp problem that is FPT with respect to the number of rejecting states:

▶ **Theorem 1.** *The* Decomp *problem for permutation DFAs is in NP. It is in FPT with parameter k, being the number of rejecting states of DFA $\mathcal{A}$, solvable in time $\mathcal{O}(2^k k^2 \cdot |\mathcal{A}|)$.*

We prove Theorem 1 by introducing the notion of *orbit-DFAs*: an orbit-DFA $\mathcal{A}^U$ of a DFA $\mathcal{A}$ is the DFA obtained by fixing a set of states U of $\mathcal{A}$ as the initial state, and letting the transition function of $\mathcal{A}$ act over it (thus the states of $\mathcal{A}^U$ are subsets of the state space of $\mathcal{A}$). We prove three key results:

- A permutation DFA is composite if and only if it can be decomposed into its orbit-DFAs (Corollary 6);
- A permutation DFA $\mathcal{A}$ can be decomposed into its orbit-DFAs if and only if for each of its rejecting states q, there exists an orbit-DFA $\mathcal{A}^U$ smaller than $\mathcal{A}$ that *covers* q, that is, one of the states of $\mathcal{A}^U$ contains q and no accepting states of $\mathcal{A}$ (Lemma 7);
- Given a permutation DFA $\mathcal{A}$ and a rejecting state q, we can determine the existence of an orbit-DFA covering q in non-deterministic time $\mathcal{O}(|\mathcal{A}|^2)$, and in deterministic time $\mathcal{O}(2^k k \cdot |\mathcal{A}|)$, where k is the number of rejecting states of $\mathcal{A}$ (Lemma 8, (apx) Algorithm 1).

These results directly imply Theorem 1. We also apply them to show that the Decomp problem is trivial for permutation DFAs with a prime number of states.

▶ **Theorem 2.** *Let $\mathcal{A}$ be a permutation DFA with at least one accepting state and one rejecting state. If the number of states of $\mathcal{A}$ is prime, then $\mathcal{A}$ is prime.*

3.1 Proof of Theorem 1

Consider a DFA $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$. We extend δ to subsets $U \subseteq Q$ in the expected way:

$$\delta(U, w) = \{q \in Q \mid q = \delta(p, w) \text{ for some } p \in U\} \text{ for every word } w \in \Sigma^*.$$

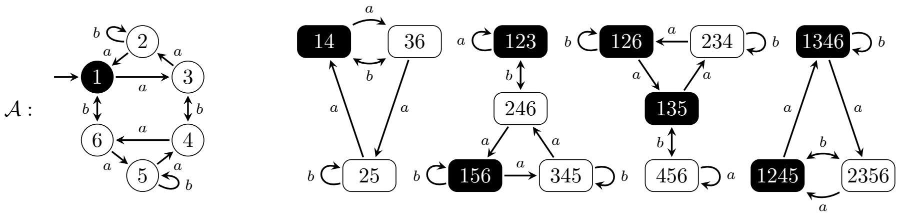


Figure 3 A DFA $\mathcal{A}$ together with some of its orbit-DFAs. Accepting states are depicted in black, an orbit-DFA can be obtained by setting a subset containing a 1 as an initial state. For instance the orbit-DFAs $\mathcal{A}^{\{1,2,3\}}$ and $\mathcal{A}^{\{1,5,6\}}$ form a decomposition of $\mathcal{A}$.

The *orbit* of U is the collection $\mathcal{C}_U = \{\delta(U,w) \subseteq Q \mid w \in \Sigma^*\}$ of subsets of Q that can be reached from U by the action of δ. If the subset $U \subseteq Q$ contains the initial state q_I of $\mathcal{A}$, we define the *orbit-DFA* $\mathcal{A}^U = \langle \Sigma, \mathcal{C}_U, U, \delta, \mathcal{C}' \rangle$, where the state space $\mathcal{C}_U$ is the orbit of U, and the set $\mathcal{C}'$ of accepting states is composed of the sets $U' \in \mathcal{C}_U$ that contain at least one of the accepting states of $\mathcal{A}$: $U' \cap F \neq \varnothing$. Note that $\mathcal{A}^U$ can alternatively be defined as the standard subset construction starting with the set $U \subseteq Q$ as initial state. The definition of the accepting states guarantees that $L(\mathcal{A}) \subseteq L(\mathcal{A}^U)$:

▶ **Proposition 3.** *Every orbit-DFA $\mathcal{A}^U$ of a DFA $\mathcal{A}$ satisfies $L(\mathcal{A}) \subseteq L(\mathcal{A}^U)$.*

Example. Let us detail the orbits of the DFA $\mathcal{A}$ depicted in Figure 3. This DFA contains six states, and generates the following non-trivial orbits on its subsets of states:

- The 15 subsets of size 2 are split into two orbits: one of size 3, and one of size 12;
- The 20 subsets of size 3 are split into three orbits: two of size 4, and one of size 12;
- The 15 subsets of size 4 are split into two orbits, one of size 3, and one of size 12.

Figure 3 illustrates the four orbits smaller than $|\mathcal{A}|$: they induce seven orbit-DFAs, obtained by setting as initial state one of the depicted subsets containing the initial state 1 of $\mathcal{A}$.

In order to prove that a DFA is composite if and only if it can be decomposed into its orbit-DFAs, we prove that every factor $\mathcal{B}$ of a permutation DFA $\mathcal{A}$ can be turned into an orbit-DFA $\mathcal{A}^U$ that is also a factor of $\mathcal{A}$, and satisfies $L(\mathcal{A}^U) \subseteq L(\mathcal{B})$. Our proof is based on a known result stating that factors can be turned into permutation DFAs:

▶ **Lemma 4** ([9, Theorem 7.4])**.** *Let $\mathcal{A}$ be a permutation DFA. For every factor $\mathcal{B}$ of $\mathcal{A}$, there exists a permutation DFA $\mathcal{C}$ satisfying $|\mathcal{C}| \leq |\mathcal{B}|$ and $L(\mathcal{A}) \subseteq L(\mathcal{C}) \subseteq L(\mathcal{B})$.*

We strengthen this result by showing how to transform factors into orbit-DFAs:

▶ **Lemma 5.** *Let $\mathcal{A}$ be a permutation DFA. For every factor $\mathcal{B}$ of $\mathcal{A}$, there exists an orbit-DFA $\mathcal{A}^U$ of A satisfying $|\mathcal{A}^U| \leq |\mathcal{B}|$ and $L(\mathcal{A}) \subseteq L(\mathcal{A}^U) \subseteq L(\mathcal{B})$.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ be a permutation DFA, and let $\mathcal{B}$ be a factor of $\mathcal{A}$. By Lemma 4, there exists a *permutation* DFA $\mathcal{B}' = \langle \Sigma, S, s_I, \eta, G \rangle$ satisfying $|\mathcal{B}'| \leq |\mathcal{B}|$ and $L(\mathcal{A}) \subseteq L(\mathcal{B}') \subseteq L(\mathcal{B})$. We build, based on $\mathcal{B}'$, an orbit-DFA $\mathcal{A}^U$ of $\mathcal{A}$ satisfying the statement.

We say that a state $q \in Q$ of $\mathcal{A}$ is *linked* to a state $s \in S$ of $\mathcal{B}'$, denoted $q \sim s$, if there exists a word $u \in \Sigma^*$ satisfying $\delta(q_I, u) = q$ and $\eta(s_I, u) = s$. Let $f : S \to 2^Q$ be the function mapping every state $s \in S$ to the set $f(s) \subseteq Q$ containing all the states $q \in Q$ that are linked to s (i.e. satisfying $q \sim s$). We set $U = f(s_I)$. In particular, the initial state q_I of $\mathcal{A}$ is in U since $\delta(q_I, \varepsilon) = q_I$ and $\eta(s_I, \varepsilon) = s_I$. We show that the orbit-DFA $\mathcal{A}^U$ satisfies the desired conditions: $|\mathcal{A}^U| \leq |\mathcal{B}'|$ and $L(\mathcal{A}) \subseteq L(\mathcal{A}^U) \subseteq L(\mathcal{B}')$.

First, we show that $|\mathcal{A}^U| \leq |\mathcal{B}'|$ by proving that the function f defined earlier maps S surjectively into the orbit of U, which is the state space of $\mathcal{A}^U$. Since both $\mathcal{A}$ and $\mathcal{B}'$ are permutation DFAs, we get that for all $q \in Q$, $s \in S$ and $a \in \Sigma$, then $q \sim s$ if and only if $\delta(q,a) \sim \eta(s,a)$ holds.[1] Therefore, for every word $v \in \Sigma^*$, $f(\eta(s_I, v)) = \delta(f(s_I), v) = \delta(U, v)$. This shows that, as required, the image of the function f is the orbit of U, and f is surjective.

To conclude, we show that $L(\mathcal{A}) \subseteq L(\mathcal{A}^U) \subseteq L(\mathcal{B}')$. Proposition 3 immediately implies that $L(\mathcal{A}) \subseteq L(\mathcal{A}^U)$. Therefore it is enough to show that $L(\mathcal{A}^U) \subseteq L(\mathcal{B}')$. Let $v \in L(\mathcal{A}^U)$. By definition of an orbit-DFA, this means that the set $\delta(U, v)$ contains an accepting state q_F of $\mathcal{A}$. Since, as stated earlier, $f(\eta(s_I, v)) = \delta(U, v)$, this implies (by definition of the function f) that the accepting state q_F of $\mathcal{A}$ is linked to $\eta(s_I, v)$, i.e., there exists a word $v' \in \Sigma^*$ such that $\delta(q_I, v') = q_F$ and $\eta(s_I, v') = \eta(s_I, v)$. Then $\delta(q_I, v') = q_F$ implies that v' is in the language of $\mathcal{A}$. Moreover, since $L(\mathcal{A}) \subseteq L(\mathcal{B}')$ by supposition, v' is also accepted by $\mathcal{B}'$, i.e., $\eta(s_I, v')$ is an accepting state of $\mathcal{B}'$. Therefore, since $\eta(q_I, v') = \eta(q_I, v)$, the word v is also in the language of $\mathcal{B}'$. This shows that $L(\mathcal{A}^U) \subseteq L(\mathcal{B}')$, which concludes the proof. ◀

As an immediate corollary, every decomposition of a permutation DFA can be transformed, factor after factor, into a decomposition into orbit-DFAs.

▶ **Corollary 6.** *A permutation DFA is composite if and only if it can be decomposed into its orbit-DFAs.*

Orbit cover. Given a rejecting state $q \in Q \setminus F$ of $\mathcal{A}$, we say that the orbit-DFA $\mathcal{A}^U$ *covers* q if $|\mathcal{A}^U| < |\mathcal{A}|$, and $\mathcal{A}^U$ contains a rejecting state $U' \subseteq Q$ that contains q. Remember that, by definition, this means that U' contains no accepting state of $\mathcal{A}$, i.e., $U' \cap F = \varnothing$. We show that permutation DFAs that can be decomposed into their orbit-DFAs are characterized by the existence of orbit-DFAs covering each of their rejecting states.

▶ **Lemma 7.** *A permutation DFA $\mathcal{A}$ is decomposable into its orbit-DFAs if and only if every rejecting state of $\mathcal{A}$ is covered by an orbit-DFA $\mathcal{A}'$ of $\mathcal{A}$ satisfying $|\mathcal{A}'| < |\mathcal{A}|$.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ be a permutation DFA. We prove both implications.

Suppose that $\mathcal{A}$ can be decomposed into its orbit-DFAs $(\mathcal{A}^{U_i})_{1 \leq i \leq k}$, and let $q \in Q \setminus F$ be a rejecting state of $\mathcal{A}$. We show that q is covered by every orbit-DFA $\mathcal{A}^{U_i}$ that rejects a word $w \in \Sigma^*$ satisfying $\delta(q_I, w) = q$. Formally, let $w \in \Sigma^*$ be a word satisfying $\delta(q_I, w) = q$. Then $w \notin L(\mathcal{A}) = \bigcap_{i=1}^n L(\mathcal{A}^{U_i})$, hence there exists $1 \leq i \leq n$ such that $w \notin L(\mathcal{A}^{U_i})$. Let $U' \subseteq Q$ be the state visited by $\mathcal{A}^{U_i}$ after reading w. Then, by applying the definition of an orbit-DFA, we get that $q \in U'$ since $\delta(q_I, w) = q$, and $U' \cap F = \varnothing$ since U' is a rejecting state of $\mathcal{A}^{U_i}$ (as $w \notin L(\mathcal{A}^{U_i})$). Therefore, $\mathcal{A}^{U_i}$ covers q. Moreover, $|\mathcal{A}^{U_i}| < |\mathcal{A}|$ since $\mathcal{A}^{U_i}$ is a factor of $\mathcal{A}$.

Conversely, let us fix an enumeration $q_1, q_2, \ldots, q_m$ of the rejecting states of $\mathcal{A}$, and suppose that for all $1 \leq i \leq m$ there is an orbit-DFA $\mathcal{A}^{U_i}$ of $\mathcal{A}$ that covers q_i and satisfies $|\mathcal{A}^{U_i}| < |\mathcal{A}|$. Let $(U_{i.j})_{1 \leq j \leq n_i}$ be an enumeration of the subsets in the orbit of U_i that contain the initial state q_I of $\mathcal{A}$. We conclude the proof by showing that $S = \{\mathcal{A}^{U_{i.j}} \mid 1 \leq i \leq m, 1 \leq j \leq n_i\}$ is a decomposition of $\mathcal{A}$. Note that we immediately get $|\mathcal{A}^{U_{i.j}}| = |\mathcal{A}^{U_i}| < |\mathcal{A}|$ for all $1 \leq i \leq m$ and $1 \leq j \leq n_i$. Moreover, Proposition 3 implies $L(\mathcal{A}) \subseteq \bigcap_{\mathcal{A}' \in S} L(\mathcal{A}')$. To complete the proof, we show that $\bigcap_{\mathcal{A}' \in S} L(\mathcal{A}') \subseteq L(\mathcal{A})$. Let $w \in \Sigma^*$ be a word rejected by $\mathcal{A}$. To prove the desired inclusion, we show that there is a DFA $\mathcal{A}' \in S$ that rejects w. Since $w \notin L(\mathcal{A})$, the

[1] Remark that for general DFAs we only get that $q \sim s$ implies $\delta(q,a) \sim \eta(s,a)$ from the determinism. It is the *backward determinism* of the permutation DFAs $\mathcal{A}$ and $\mathcal{B}'$ that gives us the reverse implication.

run of $\mathcal{A}$ on w starting from the initial state ends in a rejecting state q_i, for some $1 \leq i \leq m$. By supposition the orbit-DFA $\mathcal{A}^{U_i}$ covers q_i, hence the orbit of U_i contains a set $U' \subseteq Q$ that contains q_i and no accepting state. Note that there is no guarantee that $\mathcal{A}^{U_i}$ rejects w: while the set $\delta(U_i, w)$ contains q_i, it is not necessarily equal to U', and might contain accepting states. However, as $\mathcal{A}$ is a permutation DFA, we can reverse all of the transitions of $\mathcal{A}$ to get a path labeled by the reverse of w that starts from U' (that contains q_i), and ends in one of the sets $U_{i.j}$ (that contains q_I).[2] Therefore, by reversing this path back to normal, we get that $\delta(U_{i.j}, w) = U'$, hence the orbit-DFA $\mathcal{A}^{U_{i.j}} \in S$ rejects w. Therefore, every word rejected by $\mathcal{A}$ is rejected by an orbit-DFA $\mathcal{A}' \in S$, which shows that $\bigcap_{\mathcal{A}' \in S} L(\mathcal{A}') \subseteq L(\mathcal{A})$. ◀

This powerful lemma allows us to easily determine whether a permutation DFA is composite if we know its orbits. For instance, the DFA $\mathcal{A}$ depicted in Figure 3 is composite since the orbit-DFA $\mathcal{A}^{\{1,2,3\}}$ covers its five rejecting states. Following the proof of Lemma 7, we get that $(\mathcal{A}^{\{1,2,3\}}, \mathcal{A}^{\{1,5,6\}})$ is a decomposition of $\mathcal{A}$, and so is $(\mathcal{A}^{\{1,2,6\}}, \mathcal{A}^{\{1,3,5\}})$.

To conclude, we give an algorithm checking if a rejecting state is covered by an orbit-DFA.

▶ **Lemma 8.** *Given a permutation DFA $\mathcal{A}$ and a rejecting state q, we can determine the existence of an orbit-DFA that covers q in nondeterministic time $\mathcal{O}(k \cdot |\mathcal{A}|^2)$, and in deterministic time $\mathcal{O}(2^k k \cdot |\mathcal{A}|^2)$, where k is the number of rejecting states of $\mathcal{A}$.*

Proof. We can decide in NP whether there exists an orbit-DFA $\mathcal{A}^U$ of $\mathcal{A}$ that covers p: we non-deterministically guess among the set of rejecting states of $\mathcal{A}$ a subset U' containing p. Then, we check in polynomial time that the orbit of U' is smaller than $|\mathcal{A}|$. This property can be checked in time $\mathcal{O}(|\mathcal{A}|^2)$. Since $\mathcal{A}$ is trim, in the orbit of U' there is a set U containing the initial state of $\mathcal{A}$. Moreover, since $\mathcal{A}$ is a permutation DFA, U and U' induce the same orbit. Hence, p is covered by the orbit-DFA $\mathcal{A}^U$. Finally, we can make this algorithm deterministic by searching through the 2^k possible subsets U' of the set of rejecting states of $\mathcal{A}$. ◀

3.2 Proof of Theorem 2

Thanks to the notion of orbit DFAs we are able to prove that a permutation DFA which has a prime number of states with at least one accepting and one rejecting, is prime.

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ be a trim permutation DFA with a state space Q of prime size that contains at least one accepting state and one rejecting state. We show that the only orbit of $\mathcal{A}$ smaller than $|Q|$ is the trivial orbit $\{Q\}$. This implies that $\mathcal{A}$ cannot be decomposed into its orbit-DFAs, which proves that $\mathcal{A}$ is prime by Lemma 5.

Let us consider a strict subset $U_1 \neq \varnothing$ of the state space Q, together with its orbit $\mathcal{C}_{U_1} = \{U_1, U_2, \ldots, U_m\}$. We prove that $m \geq |Q|$. First, we show that all the U_i have the same size: since U_i is an element of the orbit of U_1, there exists a word $u_i \in \Sigma^*$ satisfying $\delta(U_1, u_i) = U_i$, and, as every word in Σ^* induces via δ a permutation on the state space, $|U_i| = |\delta(U_1, u_i)| = |U_1|$. Second, for every $q \in Q$, we define the *multiplicity* of q in $\mathcal{C}_{U_1}$ as the number $\lambda(q) \in \mathbb{N}$ of distinct elements of $\mathcal{C}_{U_1}$ containing the state q. We show that all the states q have the same multiplicity: since $\mathcal{A}$ is trim, there exists a word $u_q \in \Sigma^*$ satisfying $\delta(q_I, u_q) = q$, hence u_q induces via δ a bijection between the elements of $\mathcal{C}_{U_1}$ containing q_I and those containing q, and $\lambda(q) = \lambda(\delta(q_I, u_q)) = \lambda(q_I)$. By combining these results, we obtain $m \cdot |U_1| = \Sigma_{i=1}^{m} |U_i| = \Sigma_{q \in Q} \lambda(q) = \lambda(q_I) \cdot |Q|$. Therefore, as $|Q|$ is prime by supposition, either m or $|U_1|$ is divisible by $|Q|$. However, $U_1 \subsetneq Q$, hence $|U_1| < |Q|$, which shows that m is divisible by $|Q|$. In particular, we get $m \geq |Q|$, which concludes the proof. ◀

[2] Remark that, if $\mathcal{A}$ is not a permutation DFA, then some states might not have incoming transitions for every letter. Thus, the reversal of w might not be defined.

4 Decompositions of Commutative Permutation DFAs

We now study *commutative* permutation DFAs: a DFA $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F\rangle$ is commutative if $\delta(q, uv) = \delta(q, vu)$ for every state q and every pair of words $u, v \in \Sigma^*$. Our main contribution is an NL algorithm for the Decomp problem for commutative permutation DFAs. Moreover, we show that the complexity goes down to LOGSPACE for alphabets of fixed size.

▶ **Theorem 9.** *The Decomp problem for commutative permutation DFAs is in NL, and in LOGSPACE when the size of the alphabet is fixed.*

The proof of Theorem 9 is based on the notion of *covering word*: a word $w \in \Sigma^*$ covers a rejecting state q of a DFA $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F\rangle$ if $\delta(q, w) \neq q$, and for every $\lambda \in \mathbb{N}$, the state $\delta(q, w^\lambda)$ is rejecting. We prove two related key results:

- A commutative permutation DFA is composite if and only if each of its rejecting states is covered by a word (Lemma 12).
- We can decide in NL (LOGSPACE when the size of the alphabet is fixed) if a given rejecting state of a DFA is covered by a word (Lemma 13, and Algorithm 2 in appendix)

These results immediately imply Theorem 9. We conclude this section by showing an upper bound on the width and constructing a family of DFAs of polynomial width.

▶ **Theorem 10.** *The width of every composite permutation DFA is smaller than its size. Moreover, for all $m, n \in \mathbb{N}$ such that n is prime, there exists a commutative permutation DFA of size n^m and width $(n-1)^{m-1}$.*

We show that the width of a commutative permutation DFA is bounded by its number of rejecting states (Lemma 12). Then, for each $m, n \in \mathbb{N}$ with n prime, we define a DFA $\mathcal{A}_n^m$ of size n^m that can be decomposed into $(n-1)^{m-1}$ factors (Proposition 14), but not into $(n-1)^{m-1} - 1$ (Proposition 15).

4.1 Proof of Theorem 9

The proof is based on the following key property of commutative permutation DFAs: In a permutation DFA $\mathcal{A}$, every input word acts as a permutation on the set of states, generating disjoint cycles, and if $\mathcal{A}$ is commutative these cycles form an orbit.

▶ **Proposition 11.** *Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F\rangle$ be a commutative permutation DFA. For all $u \in \Sigma^*$, the sets $(\{\delta(q, u^\lambda) \mid \lambda \in \mathbb{N}\})_{q \in Q}$ partition Q and form an orbit of $\mathcal{A}$.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F\rangle$ be a commutative permutation DFA. Given $u \in \Sigma^*$ and $q \in Q$, the sequence of states $\delta(q, u), \delta(q, u^2), \dots, \delta(q, u^i)$ visited by applying δ on iterations of u eventually repeats i.e. $\delta(q, u^x) = \delta(q, u^y) = p$ for some $x, y \in \mathbb{N}$ and $p \in Q$. Since $\mathcal{A}$ is a permutation DFA, it is both forward and backward deterministic, thus the set of visited states $\{\delta(q, u^\lambda) \mid \lambda \in \mathbb{N}\}$ is a cycle that contain both p and q. The collection $(\{\delta(q, u^\lambda) \mid \lambda \in \mathbb{N}\})_{q \in Q}$ forms an orbit of $\mathcal{A}$ by commutativity. Formally, for all $u, v \in \Sigma^*$ and every $q \in Q$, we have: $\delta(\{\delta(q, u^\lambda) | \lambda \in \mathbb{N}\}, v) = \{\delta(q, u^\lambda v) | \lambda \in \mathbb{N}\} = \{\delta(q, vu^\lambda) | \lambda \in \mathbb{N}\} = \{\delta(\delta(q, v), u^\lambda) | \lambda \in \mathbb{N}\}$. ◀

We proved with Corollary 6 and Lemma 7 that a permutation DFA is composite if and only if each of its rejecting states is covered by an orbit-DFA. We now reinforce this result for *commutative* permutation DFAs. As stated before, we say that a word $u \in \Sigma^*$ *covers* a rejecting state q of a DFA $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F\rangle$ if u induces from q a non-trivial cycle composed of rejecting states: $\delta(q, u) \neq q$, and $\delta(q, u^\lambda)$ is rejecting for all $\lambda \in \mathbb{N}$. Note that the collection $(\{\delta(q, u^\lambda) \mid \lambda \in \mathbb{N}\})_{q \in Q}$ forms an orbit of $\mathcal{A}$ by Proposition 11. We show that we can determine if $\mathcal{A}$ is composite by looking for words covering its rejecting states.

▶ **Lemma 12.** *For every $k \in \mathbb{N}$, a commutative permutation DFA $\mathcal{A}$ is k-factor composite if and only if there exist k words that, together, cover all the rejecting states of $\mathcal{A}$.*

Proof. Let $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ be a commutative permutation DFA and $k \in \mathbb{N}$. We start by constructing k factors based on k covering words. Suppose that there exist k words $u_1, u_2, \dots, u_k$ such that every rejecting state $q \in Q \setminus F$ is covered by one of the u_i. Note that all the u_i covering at least one state q do not act as the identity on Q (since $\delta(q, u_i) \neq q$), therefore we suppose, without loss of generality, that none of the u_i acts as the identity on Q. For every $1 \leq i \leq k$, let $U_i = \{\delta(q_I, u_i^\lambda) \mid \lambda \in \mathbb{N}\}$. We show that $(\mathcal{A}^{U_i})_{1 \leq i \leq k}$ is a decomposition of $\mathcal{A}$. As none of the u_i acts as the identity on Q, Proposition 11 implies that every $\mathcal{A}^{U_i}$ is smaller than $\mathcal{A}$. Moreover, Proposition 3 implies that $L(\mathcal{A}) \subseteq L(\mathcal{A}^{U_i})$, hence $L(\mathcal{A}) \subseteq \bigcap_{j=1}^k L(\mathcal{A}^{U_j})$. To conclude, we show that $\bigcap_{j=1}^k L(\mathcal{A}^{U_j}) \subseteq L(\mathcal{A})$. Let $u \notin L(\mathcal{A})$. By supposition, there exists $1 \leq i \leq k$ such that u_i covers $\delta(q_I, u)$. As a consequence, the set

$$\begin{aligned} \delta(U_i, u) &= \delta(\{\delta(q_I, u_i^\lambda) \mid \lambda \in \mathbb{N}\}, u) = \{\delta(q_I, u_i^\lambda u) \mid \lambda \in \mathbb{N}\} = \{\delta(q_I, u u_i^\lambda) \mid \lambda \in \mathbb{N}\} \\ &= \{\delta(\delta(q_I, u), u_i^\lambda) \mid \lambda \in \mathbb{N}\} \end{aligned}$$

contains no accepting state of $\mathcal{A}$, hence it is a rejecting state of $\mathcal{A}^{U_i}$. As a consequence, we get $u \notin L(\mathcal{A}^{U_i}) \supseteq \bigcap_{j=1}^k L(\mathcal{A}^{U_j})$, which proves that $\bigcap_{j=1}^k L(\mathcal{A}^{U_j}) \subseteq L(\mathcal{A})$.

We now construct k covering words based on k factors. Suppose that $\mathcal{A}$ has a k-factor decomposition $(\mathcal{B}_i)_{1 \leq i \leq k}$. Lemma 4 directly implies that this decomposition can be transformed into a decomposition $(\mathcal{C}_i)_{1 \leq i \leq k}$ of $\mathcal{A}$, where $\mathcal{C}_i = \langle \Sigma, S_i, s_I^i, \eta_i, G_i \rangle$ are permutation DFAs. For every $1 \leq i \leq k$, we build a word u_i based on $\mathcal{C}_i$, we prove that every rejecting state of $\mathcal{A}$ is covered by one of these u_i. Consider $1 \leq i \leq k$. Since $\mathcal{C}_i$ is a factor of $\mathcal{A}$, in particular $|\mathcal{C}_i| < |\mathcal{A}|$, hence there exist two input words $v_i, w_i \in \Sigma^*$ such that $\mathcal{A}$ reaches different states on v_i and w_i, but $\mathcal{C}_i$ reaches the same state: $\delta(q_I, v_i) \neq \delta(q_I, w_i)$ but $\eta_i(s_I^i, v_i) = \eta_i(s_I^i, w_i)$. Note that both $\mathcal{A}$ and $\mathcal{C}_i$ are permutation DFAs, hence there exists a power $v_i^{\kappa_i}$ of v_i that induces the identity function on both state spaces Q and S_i. We set $u_i = w_i v_i^{\kappa_i - 1}$, which guarantees that:

$$\begin{aligned} &\delta(q_I, u_i) = \delta(\delta(q_I, w_i), v_i^{\kappa_i - 1}) \neq \delta(\delta(q_I, v_i), v_i^{\kappa_i - 1}) = \delta(q_I, v_i^{\kappa_i}) = q_I; \\ &\eta_i(s_I^i, u_i) = \eta_i(\eta_i(s_I^i, w_i), v_i^{\kappa_i - 1}) = \eta_i(\eta_i(s_I^i, v_i), v_i^{\kappa_i - 1}) = \eta_i(s_I^i, v_i^{\kappa_i}) = s_I^i. \end{aligned}$$

In other words, u_i moves the initial state q_I of $\mathcal{A}$, but fixes the initial state s_I^i of $\mathcal{C}_i$.

We now prove that each rejecting state of $\mathcal{A}$ is covered by one of the u_i. Let $q \in Q \setminus F$ be a rejecting state of $\mathcal{A}$. Since $\mathcal{A}$ is trim, there exists a word $u_q \in \Sigma^*$ such that $\delta(q_I, u_q) = q$. Then, as $u_q \notin L(\mathcal{A})$ and $(\mathcal{C}_i)_{1 \leq i \leq k}$ is a decomposition of $\mathcal{A}$, there exists $1 \leq i \leq k$ such that $u_q \notin L(\mathcal{C}_i)$. We show that the word u_i covers the rejecting state q: we prove that $\delta(q, u_i) \neq q$, and that $\delta(q, u_i^\lambda)$ is rejecting for every $\lambda \in \mathbb{N}$. First, since $\mathcal{A}$ is a commutative permutation DFA and u_i moves q_I, we get that $\delta(q, u_i) = \delta(q_I, u_q u_i) = \delta(q_I, u_i u_q) \neq \delta(q_I, u_q) = q$. Moreover, for all $\lambda \in \mathbb{N}$, Since $u_q \notin L(\mathcal{C}_i)$ by supposition and u_i fixes s_I^i, the DFA $\mathcal{C}_i$ also rejects the word $u_i^\lambda u_q$. Therefore, as $L(\mathcal{A}) \subseteq L(\mathcal{C}_i)$, we finally get that $\delta(q, u_i^\lambda) = \delta(q_I, u_q u_i^\lambda) = \delta(q_I, u_i^\lambda u_q)$ is a rejecting state of $\mathcal{A}$. ◀

By Lemma 12, to conclude the proof of Theorem 9 we show that we can decide in NL (and in LOGSPACE when the size of the alphabet is fixed) whether a given rejecting state of a DFA is covered by a word (since in the DECOMP problem we can afford to pick a covering word for each state). As we consider commutative permutation DFAs, we can represent a covering word by the number of occurrences of each letter, which are all bounded by $|Q|$.

▶ **Lemma 13.** *Let $\mathcal{A}$ be a commutative permutation DFA and p a rejecting state.*

1. *We can determine the existence of a word covering p in space $\mathcal{O}(|\Sigma| \cdot \log |Q|)$;*
2. *We can determine the existence of a word covering p in NL;*

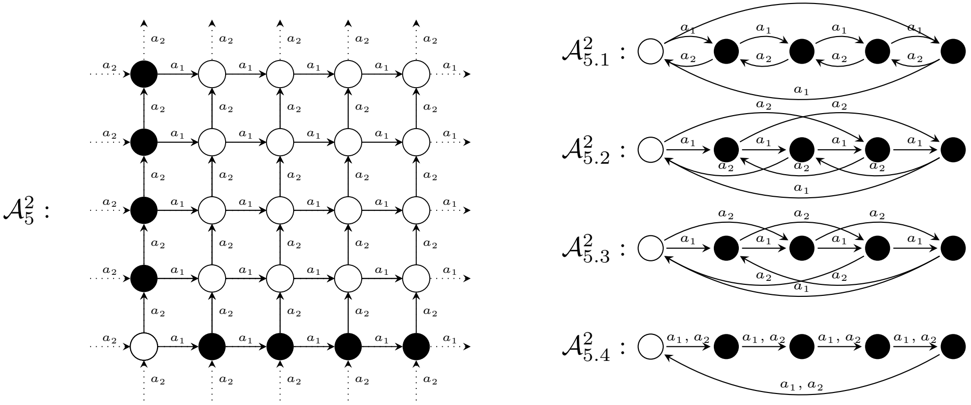


Figure 4 The DFA $\mathcal{A}_5^2$ recognising the language L_5^2, together with its decomposition into four non-trivial orbit-DFAs. Final states are depicted in black.

4.2 Proof of Theorem 10

As a direct consequence of Lemma 12, the width of every commutative permutation DFA $\mathcal{A}$ is bounded by the number of rejecting states of $\mathcal{A}$, hence, it is smaller than $|\mathcal{A}|$. To conclude the proof of Theorem 10, for all $m, n \in \mathbb{N}$ with n prime, we define a DFA $\mathcal{A}_n^m$ of size n^m and width $(n-1)^{m-1}$ on the alphabet $\Sigma = \{a_1, a_2, \ldots, a_m\}$. For all $\ell \in \mathbb{N}$, let $[\ell]$ denote the equivalence class of ℓ modulo n. Let $L_n^m \subseteq \Sigma^*$ be the language composed of the words w such that for at least one letter $a_i \in \Sigma$ the number $\#_{a_i}(w)$ of a_i in w is a multiple of n, and for at least one (other) letter $a_j \in \Sigma$, the number $\#_{a_j}(w)$ of a_j in w is *not* a multiple of n:

$$L_n^m = \{w \in \Sigma^* \mid [\#_{a_i}(w)] = [0] \text{ and } [\#_{a_j}(w)] \neq [0] \text{ for some } 1 \leq i, j \leq m\}.$$

The language L_n^m is recognised by a DFA $\mathcal{A}_n^m$ of size n^m that keeps track of the value modulo n of the number of each a_i already processed. The state space of $\mathcal{A}_n^m$ is the direct product $(\mathbb{Z}/n\mathbb{Z})^m$ of m copies of the cyclic group $\mathbb{Z}/n\mathbb{Z} = ([0], [1], \ldots, [n-1])$; the initial state is $([0], [0], \ldots, [0])$; the final states are the ones containing at least one component equal to $[0]$ and one component distinct from $[0]$; and the transition function increments the i^{th} component when an a_i is read: $\delta(([j_1], [j_2], \ldots, [j_m]), a_i) = ([j_1], [j_2], \ldots, [j_{i-1}], [j_i + 1], [j_{i+1}], \ldots, [j_m])$. Figure 4 illustrates the particular case $n = 5$ and $m = 2$.

To prove that the width of $\mathcal{A}_n^m$ is $(n-1)^{m-1}$, we first show that the $(n-1)^{m-1}$ words $\{a_1 a_2^{\lambda_2} \ldots a_m^{\lambda_m} \mid 1 \leq \lambda_i \leq n-1\}$ cover all the rejecting states, thus by Lemma 12:

▶ **Proposition 14.** *The DFA $\mathcal{A}_n^m$ is $(n-1)^{m-1}$-factor composite.*

Then, we prove that there exist no word that covers two states among the $(n-1)^{m-1}$ rejecting states $\{([1], [k_2], [k_3], \ldots, [k_m]) \mid 1 \leq k_i \leq m-1\}$. Therefore, we need at least $(n-1)^{m-1}$ words to cover all of the states, thus by Lemma 12:

▶ **Proposition 15.** *The DFA $\mathcal{A}_n^m$ is not $((n-1)^{m-1} - 1)$-factor composite.*

5 Bounded Decomposition

We finally study the Bound-Decomp problem: Given a DFA $\mathcal{A}$ and an integer $k \in \mathbb{N}$ encoded in unary, can we determine whether $\mathcal{A}$ is decomposable into k factors? For the general setting, we show that the problem is in PSPACE: it can be solved by non-deterministically guessing k factors, and checking that they form a decomposition.

▶ **Theorem 16.** *The* Bound-Decomp *problem is in* PSPACE.

For commutative permutation DFAs, we obtain a better algorithm through the use of the results obtained in the previous sections, and we show a matching hardness result.

▶ **Theorem 17.** *The* Bound-Decomp *problem for commutative permutation* DFAs *is* NP-*complete.*

Both parts of the proof of Theorem 17 are based on Lemma 12: a commutative permutation DFA is k-factor composite if and only if there exist k words covering all of its rejecting states. We prove the two following results:

- Bounded compositionnality is decidable in NP, as it is sufficient to non-deterministically guess a set of k words, and check whether they cover all rejecting states (Lemma 19);
- The NP-hardness is obtained by reducing the Hitting Set problem, a well known NP-complete decision problem. We show that searching for k words that cover the rejecting states of a DFA is as complicated as searching for a hitting set of size k (Lemma 20).

We finally give a LOGSPACE algorithm based on known results for DFAs on unary alphabets [7].

▶ **Theorem 18.** *The* Bound-Decomp *problem for unary* DFAs *is in* LOGSPACE.

Sketch. Recall that a unary DFA $\mathcal{A} = \langle \{a\}, Q, q_I, \delta, F \rangle$ consists of a chain of states leading into one cycle of states. The case where the chain is non-empty is considered in Lemmas 8 and 10 of [7]. We prove that the criteria of these lemmas can be checked in LOGSPACE. If the chain of $\mathcal{A}$ is empty, then $\mathcal{A}$ is actually a commutative permutation DFA. In this case, by Proposition 11 for every word $u = a^i \in \{a\}^*$, the orbit of the set $\{\delta(q_I, u^\lambda) \mid \lambda \in \mathbb{N}\}$ is a partition ρ on Q, and every set in ρ has the same size s_ρ. Both s_ρ and $|\rho|$ divide $|Q|$. For $u = a^i$ where i and $|Q|$ are co-prime, the induced orbit DFA has a single state and thus cannot be a factor of $\mathcal{A}$. Further, if $i_1 < |Q|$ divides $i_2 < |Q|$, then all states covered by a^{i_1} are also covered by a^{i_2}. Hence, w.l.o.g., we only consider words of the form a^i where i is a *maximal divisor* of $|Q|$ in order to generate orbit-DFAs of $\mathcal{A}$ that are candidates for the decomposition. Now, let $p_1^{j_1} \cdot p_2^{j_2} \cdot \ldots \cdot p_m^{j_m} = |Q|$ be the prime factor decomposition of $|Q|$. By Lemma 12 we have that $\mathcal{A}$ is k-factor composite if and only if a selection of k words from the set $\mathcal{W} = \{a^{|Q|/p_i} \mid 1 \leq i \leq m\}$ cover all the rejecting states of $\mathcal{A}$. As $|\mathcal{W}| = m$ is logarithmic in $|Q|$, we can iterate over all sets in $2^{\mathcal{W}}$ of size at most k in LOGSPACE using a binary string indicating the characteristic function. By Lemma 13, checking whether a state $q \in Q$ is covered by the current collection of k words can also be done in LOGSPACE. ◀

5.1 Proof of Theorem 17

By Lemma 12, a commutative permutation DFA $\mathcal{A}$ is k-factor composite if and only if its rejecting states can be covered by k words. As we can suppose that covering words have size linear in $|\mathcal{A}|$ (see proof of Lemma 13), the Bound-Decomp problem is decidable in NP: we guess a set of k covering words and check in polynomial time if they cover all rejecting states.

▶ **Lemma 19.** *The* Bound-Decomp *problem for commutative permutation* DFAs *is in* NP.

We show that the problem is NP-hard by a reduction from the Hitting Set problem.

▶ **Lemma 20.** *The* Bound-Decomp *problem is* NP-*hard for commutative permutation* DFAs.

Proof. The proof goes by a reduction from the Hitting Set problem (HIT for short), known to be NP-complete [4]. The HIT problem asks, given a finite set $S = \{1, 2, \ldots, n\} \subseteq \mathbb{N}$, a finite collection of subsets $\mathcal{F} = \{C_1, C_2, \ldots, C_m\} \subseteq 2^S$, and an integer $k \in \mathbb{N}$, whether there

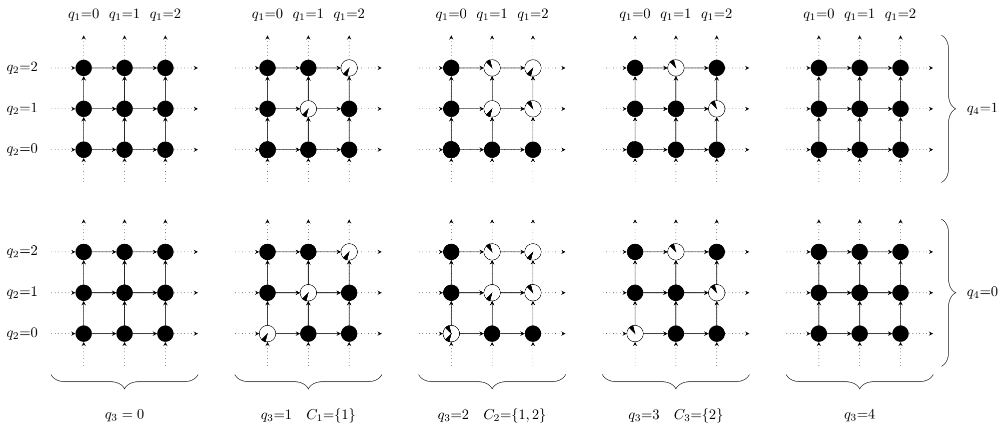


Figure 5 DFA representing the instance of HIT with $S = \{1, 2\}$ and $\mathcal{F} = \{\{1\}, \{1, 2\}, \{2\}\}$ using $\mu = 3$ and $\tau = 5$. Accepting states are filled black while rejecting states are sectored.

is a subset $X \subseteq S$ with $|X| \leq k$ and $X \cap C_i \neq \varnothing$ for all $1 \leq i \leq m$. We describe how to construct a DFA $\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F\rangle$ that is $(k+1)$-factor composite if and only if the HIT instance $\langle S, \mathcal{F}, k\rangle$ has a solution.

Automaton construction. To be constructed, the automaton $\mathcal{A}$ requires μ, τ defined as the smallest prime numbers that fulfill $n < \mu$ and $m < \tau$ and $2 < \mu < \tau$. By Bertrand's postulate [11], μ and τ have a value polynomial in $m + n$. The state space of $\mathcal{A}$ is defined as $Q = \{0, 1, \ldots, \mu - 1\} \times \{0, 1, \ldots, \mu - 1\} \times \{0, 1, \ldots, \tau - 1\} \times \{0, 1\}$ with $q_I = (0, 0, 0, 0)$ as initial state. Let us define the subset of states $Q_\perp = \{(q_1, q_2, q_3, q_4) \in Q \mid q_4 = 0\}$ to encode instances of HIT and the subset $Q_\top = \{(q_1, q_2, q_3, q_4) \in Q \mid q_4 = 1\}$ which is a copy of $Q_\perp$ with minor changes. The example in Figure 5 gives some intuition on the construction of $\mathcal{A}$. The DFA $\mathcal{A}$ is defined over the alphabet $\Sigma = \{a, b, c, d\}$ with the transition function defined for each state $q = (q_1, q_2, q_3, q_4)$ by $\delta(q, a) = (q_1 + 1 \mod \mu, q_2, q_3, q_4)$, $\delta(q, b) = (q_1, q_2 + 1 \mod \mu, q_3, q_4)$, $\delta(q, c) = (q_1, q_2, q_3 + 1 \mod \tau, q_4)$ and $\delta(q, d) = (q_1, q_2, q_3, q_4 + 1 \mod 2)$. Note that, $\mathcal{A}$ can be seen as a product of four prime finite fields. In particular, for every $q_3 \in \{0, \ldots, \tau - 1\}$ the subset of states $\{(x, y, q_3, 0) \in Q_\perp \mid 0 \leq x, y \leq \mu - 1\}$ can be seen as the direct product of two copies of the field of order μ (a.k.a. $\mathbb{F}_\mu$), thus inheriting the structure of a $\mathbb{F}_\mu$-vector space of origin $(0, 0, q_3, 0)$. We use these τ disjoint vector spaces to represent the collections of $\mathcal{F}$ thanks to the acceptance of states. More precisely, each collection $C_i \in \mathcal{F}$ is encoded through the vector space $\{(x, y, i, 0) \in Q_\perp \mid 1 \leq i \leq m\}$ and each $v \in C_i$ is encoded by the non-acceptance of all states belonging to the line $\{(x, y, i, 0) \in Q_\perp \mid y = vx \mod \mu\}$. In Figure 5, each C_i is presented by an instance of $\mathbb{F}_3 \times \mathbb{F}_3$ and each $v \in \mathcal{C}_i$ is depicted by rejecting states with the same emphasized sector. Since $\tau > m$, there are extra vector spaces for which all states are accepting i.e. $\{(q_1, q_2, q_3, 0) \in Q_\perp \mid q_3 \notin \{1, 2, \ldots, m\}\} \subseteq F$. The acceptance of states of $Q_\top$ is defined similarly as for $Q_\perp$ except that the origins of vector spaces are accepting in $Q_\top$ (see Figure 5). Formally, the rejecting states of $\mathcal{A}$ is defined by $\overline{F} = R_\perp \cup R_\top$ where $R_\perp = \{(q_1, q_2, q_3, 0) \in Q_\perp \mid q_2 = vq_1 \mod \mu, 1 \leq q_3 \leq m, v \in C_{q_3}\}$ and $R_\top = \{(q_1, q_2, q_3, 1) \in Q_\top \mid (q_1, q_2, q_3, 0) \in R_\perp, q_1 \neq 0, q_2 \neq 0\}$. All other states are accepting, i.e., we set $F = Q \setminus \overline{F}$. So, the acceptance of the subsets of states $Q_\perp$ and $Q_\top$ only differ by $O \cap Q_\perp \subseteq \overline{F}$ and $O \cap Q_\top \subseteq F$ where $O = \{(0, 0, q_3, q_4) \in Q \mid q_3 \in \{1, \ldots, m\}\}$.

The cornerstone which holds the connection between the two problems is the way the rejecting states of O can be covered. In fact, since $Q_\top$ mimics $Q_\perp$ for states in $Q \setminus O$, all rejecting states of $Q \setminus O$ can be covered by the single word $d \in \Sigma$. In addition, most words do not cover any rejecting states of $\mathcal{A}$, as stated by the following claim. Hereafter, we say that a word $w \in \Sigma^*$ is *concise* when it satisfies $\#_\sigma(w) < h_\sigma$ for all $\sigma \in \Sigma$, where $h_\sigma \in \{2, \mu, \tau\}$ is the size of the cycle induced by σ.

▷ Claim 21. Let $u \in \Sigma^*$ be a concise word that covers some rejecting state of $\mathcal{A}$:

1. u must belong either in $\{d\}^*$ or in $\{a, b\}^* \setminus (\{a\}^* \cup \{b\}^*)$.
2. u covers some rejecting state of $Q_\top$ iff u covers *all* rejecting states of $Q_\top$ iff $u = d$.
3. u covers $(0, 0, i, 0) \in O$ iff $u \in \{a, b\}^*$ and $\#_b(u) \equiv v \cdot \#_a(u) \mod \mu$ for some $v \in C_i$.

Proof of Item 1. The statement is a direct consequence of the following:
i. Every concise word u satisfying $\#_c(u) > 0$ covers no rejecting state of $\mathcal{A}$;
ii. Every concise word $u \in \{a\}^* \cup \{b\}^*$ covers no rejecting state of $\mathcal{A}$;
iii. Every concise word u satisfying $\#_a(u) > 0$ and $\#_d(u) > 0$ covers no rejecting state of $\mathcal{A}$;
iv. Every concise word u satisfying $\#_b(u) > 0$ and $\#_d(u) > 0$ covers no rejecting state of $\mathcal{A}$.
In order to prove these four properties, we now fix a state $q = (q_1, q_2, q_3, q_4) \in Q$, and we show that, in each case, iterating a word of the corresponding form starting from q will eventually lead to an accepting state:

(i.) Let u be a concise word satisfying $\#_c(u) > 0$. Since u is concise we have $\#_c(u) < \tau$. Hence, as τ is prime, there exists $\lambda \in \mathbb{N}$ such that $\lambda \cdot \#_c(u) \equiv -q_3 \mod \tau$. Therefore the third component of $\delta(q, u^\lambda)$ is 0, thus it is an accepting state of $\mathcal{A}$.

(ii.) Let $u \in \{a\}^*$ be a concise word (if $u \in \{b\}^*$ instead, the same proof works by swapping the roles of q_1 and q_2). Since u is concise we have $0 < \#_a(u) < \mu$. Hence, as μ is prime there exists $\lambda_1, \lambda_2 \in \mathbb{N}$ satisfying $\lambda_1 \cdot \#_a(u) \equiv -q_1 \mod \mu$ and $\lambda_2 \cdot \#_a(u) \equiv -q_1 + 1 \mod \mu$. Therefore, if $q_2 \neq 0$, we get that $\delta(q, u^{\lambda_1}) = (0, q_2, q_3, q_4)$ is an accepting state of $\mathcal{A}$, and if $q_2 = 0$, we get that $\delta(q, u^{\lambda_2}) = (1, 0, q_3, q_4)$ is an accepting state of $\mathcal{A}$.

(iii.) Let u be a concise word satisfying $\#_a(u) > 0$ and $\#_d(u) > 0$. Since μ is a prime number greater than 2, there exist $\alpha \in \mathbb{N}$ such that $\mu - 2\alpha = 1$, thus $2\alpha \equiv -1 \mod \mu$. Moreover, since u is concise we have $\#_d(u) = 1$ and $\#_a(u) < \mu$. Hence there exists $\beta \in \mathbb{N}$ such that $\beta \cdot \#_a(u) \equiv 1 \mod \mu$. Therefore, if we let $\lambda = 2\alpha\beta q_1 + \mu(1 - p_4)$, we get

$$\#_a(u^\lambda) = 2\alpha \cdot \beta\#_a(u) \cdot q_1 + \mu(1 - p_4) \cdot \#_a(u) \equiv -q_1 \mod \mu;$$
$$\#_d(u^\lambda) = 2\alpha\beta q_1 + \mu \cdot (1 - p_4) \equiv 1 - p_4 \mod 2;$$

As a consequence, the first component of $\delta(q, u^\lambda)$ is 0 and its fourth component is 1, hence it is an accepting state of $\mathcal{A}$.

(iv.) Let u be a concise word satisfying $\#_b(u) > 0$ and $\#_d(u) > 0$. Then we can prove that u does not cover q as in point (3), by swapping the roles of q_1 and q_2. ◁

Proof of Item 2. First, remark that d is the only concise word of $\{d\}^*$. By construction of $\mathcal{A}$, we have $(q_1, q_2, q_3, 0) \in F$ if and only if $(q_1, q_2, q_3, 1) \in F$ holds for all $(q_1, q_2, q_3, q_4) \in Q \setminus O$. Thus, for all $(q_1, q_2, q_3, q_4) \in \overline{F} \setminus O$ we have

$$\{\delta((q_1, q_2, q_3, q_4), d^\lambda) \mid \lambda \in \mathbb{N}\} = \{(q_1, q_2, q_3, x) \mid x \in \{0, 1\}\} \subseteq \overline{F}.$$

Hence, if $u = d$ then u covers all rejecting states of of $Q_\top$.

Now suppose that $u \in \Sigma^*$ covers some rejecting state $q = (q_1, q_2, q_3, 1) \in Q_\top$. By Item (1.), either $u \in \{d\}^*$ or $u \in \{a, b\}^* \setminus (\{a\}^* \cup \{b\}^*)$. We show that $u \in \{d\}^*$, by supposing that $\#_a(u) > 0$ and deriving a contradiction. Since μ is prime, there exists $\lambda \in \mathbb{N}$ satisfying $\lambda \cdot \#_a(u) \equiv -q_1 \mod \mu$. Therefore the first component of $\delta(q, u^\lambda)$ is 0 and its fourth component is 1, hence it is accepting, which contradicts the assumption that u covers q. ◁

Proof of Item 3. Consider a rejecting state $q = (0,0,i,0) \in O$. First, remark that no word in $\{d\}^*$ covers q since $(0,0,i,1)$ is accepting. Therefore, by Item (1.), the only concise words that can cover q are the words $u \in \{a,b\}^* \setminus (\{a\}^* \cup \{b\}^*)$. For such a word u, since μ is prime, by Bezout's identity there exists $0 < v < \mu$ satisfying $\#_b(x) \equiv v \cdot \alpha \#_a(x) \mod \mu$, hence

$$\{\delta((0,0,i,0),u^\lambda) \mid \lambda \in \mathbb{N}\} = \{(q_1,q_2,i,0) \in Q \mid q_2 \equiv vq_1 \mod \mu\}.$$

If $v \in C_i$, all the states in this set are rejecting, thus u covers $(0,0,i,0)$, but if $v \notin C_i$, all these states except from $(0,0,i,0)$ are accepting, thus u does not cover $(0,0,i,0)$. ◁

We finally conclude the proof of Lemma 20 by proving that the sets of the initial instance of HIT are hitting if and only if the automaton $\mathcal{A}$ is composite.

If sets are hitting then the automaton is composite. Thanks to Lemma 12, we can show that $\mathcal{A}$ is $(k+1)$-factor composite by finding $(k+1)$ words, namely $w_\top, w_1, w_2, \ldots, w_k$, which all together cover all the rejecting states of $\mathcal{A}$. From the HIT solution $X = \{v_1, v_2, \ldots, v_k\} \subseteq S$, we define $w_j = ab^{v_j}$ for all $1 \leq j \leq k$. We prove now that for all $1 \leq i \leq m$, the rejecting state $(0,0,i,0) \in O$ is covered by some w_j. Since $X \cap C_i \neq \varnothing$, there exists $v_j \in X \cap C_i$. Moreover, by definition of w_j, we have $w_j \in \{a,b\}^*$ and $\#_b(w_j) \equiv v_j \cdot \#_a(w_j) \mod \mu$. Therefore, by Claim 21.3, $(0,0,i,0)$ is covered by w_j. Finally, we take $w_\top = d$ which covers all rejecting states $\overline{F} \setminus O$ by Claim 21.2.

If the automaton is composite then the sets are hitting. Suppose that $\mathcal{A}$ is $(k+1)$-factor composite. Hence, by Lemma 12, there exists a set W of at most $k+1$ words such that all rejecting states of $\mathcal{A}$ can be covered by some $w \in W$. In addition, we assume that each $w \in W$ is concise: if this is not the case, we can remove the superfluous letter to obtain a concise words that cover the same rejecting states. As a consequence of Claim 21.2, to cover the rejecting states of $Q_\top$, the set W needs the word d, thus W contains at most k words in $\{a,b\}^*$. Moreover, by Claim 21.3, for every $1 \leq i \leq m$, to cover $(0,0,i,0) \in O$ the set W needs a word $u_i \in \{a,b\}^*$ satisfying $\#_b(u_i) \equiv v_i \cdot \#_a(u_i) \mod \mu$ for some $v_i \in C_i$. To conclude, we construct $X = \{v_i \mid 1 \leq i \leq m\}$ which is a solution since $|X| \leq k$ due to $W \cap \{d\}^* \neq \varnothing$, and for each $C \in \mathcal{F}$ we have $X \cap C \neq \varnothing$. ◀

6 Discussion

We introduced in this work powerful techniques to treat the DECOMP problem for permutation DFAs. We discuss how they could help solving the related questions that remain open:

- How do the insights obtained by our results translate to the general setting?
- How can we use our techniques to treat other variants of the DECOMP problem?

Solving the general setting. The techniques presented in this paper rely heavily on the group structure of transition monoids of permutation DFAs, thus cannot be used directly in the general setting. They still raise interesting questions: Can we also obtain an FPT algorithm with respect to the number of rejecting states in the general setting? Some known results point that bounding the number of states is not as useful in general as it is for permutation DFAs: while it is known that every permutation DFA with a single rejecting state is prime [9], there exist (non-permutation) DFAs with a single rejecting state that are composite. However, we still have hope to find a way to adapt our techniques: maybe, instead of trying to cover rejecting *states*, we need to cover rejecting *behaviours* of the transition monoid. Another way to improve the complexity in the general setting would be to bound the *width* of DFAs: we defined here a family of DFAs with polynomial width, do there exist families with exponential width? If this is not the case (i.e., every composite DFA has polynomial width), we would immediately obtain a PSPACE algorithm for the general setting.

Variants of the Decomp problem. In this work, we focused on the Bound-Decomp problem, that limits the *number* of factors in the decompositions. Numerous other restrictions can be considered. For instance, the Fragmentation problem bounds the *size* of the factors: Given a DFA $\mathcal{A}$ and $k \in \mathbb{N}$, can we decompose $\mathcal{A}$ into DFAs of size smaller than k? Another interesting restriction is proposed by the Compression problem, that proposes a trade-off between limiting the size and the number of the factors: given a DFA $\mathcal{A}$, can we decompose $\mathcal{A}$ into DFAs $(\mathcal{A}_i)_{1 \leq i \leq k}$ satisfying $\Sigma_{i=1}^{n} |\mathcal{A}_i| < |\mathcal{A}|$? How do these problems compare to the ones we studied? We currently conjecture that the complexity of the Fragmentation problem matches the Decomp problem, while the complexity of the Compression problem matches the Bound-Decomp problem: for commutative permutation DFAs, the complexity seems to spike precisely when we limit the number of factors.

References

1 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking.* MIT Press, 2008.

2 Edmund M. Clarke, David E. Long, and Kenneth L. McMillan. A language for compositional specification and verification of finite state hardware controllers. *Proceedings of the IEEE*, 79(9):1283–1292, 1991. doi:10.1109/5.97298.

3 Willem P. de Roever, Hans Langmaack, and Amir Pnueli, editors. *Compositionality: The Significant Difference, International Symposium, COMPOS'97, Bad Malente, Germany, September 8-12, 1997. Revised Lectures*, volume 1536 of *Lecture Notes in Computer Science*. Springer, 1998. doi:10.1007/3-540-49213-5.

4 Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., USA, 1979.

5 Stephen Gould, Ernest Peltzer, Robert Matthew Barrie, Michael Flanagan, and Darren Williams. Apparatus and method for large hardware finite state machine with embedded equivalence classes, 2007. US Patent 7,180,328.

6 G. H. Hardy. An introduction to the theory of numbers. *Bulletin of the American Mathematical Society*, 35(6):778–818, November 1929. URL: https://projecteuclid.org:443/euclid.bams/1183493592.

7 Ismaël Jecker, Orna Kupferman, and Nicolas Mazzocchi. Unary prime languages. In Javier Esparza and Daniel Král, editors, *45th International Symposium on Mathematical Foundations of Computer Science, MFCS 2020, August 24-28, 2020, Prague, Czech Republic*, volume 170 of *LIPIcs*, pages 51:1–51:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.51.

8 Michal Kunc and Alexander Okhotin. Reversibility of computations in graph-walking automata. In Krishnendu Chatterjee and Jirí Sgall, editors, *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, volume 8087 of *Lecture Notes in Computer Science*, pages 595–606. Springer, 2013. doi:10.1007/978-3-642-40313-2_53.

9 Orna Kupferman and Jonathan Mosheiff. Prime languages. *Inf. Comput.*, 240:90–107, 2015. doi:10.1016/j.ic.2014.09.010.

10 Rolf Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5(3):183–191, 1961. doi:10.1147/rd.53.0183.

11 Jaban Meher and M Ram Murty. Ramanujan's proof of Bertrand's postulate. *The American Mathematical Monthly*, 120(7):650–653, 2013. doi:10.4169/amer.math.monthly.120.07.650.

12 Alon Netser. Decomposition of safe languages. Amirim Research Project report from the Hebrew University, 2018.

13 Volnei A. Pedroni. *Finite State Machines in Hardware: Theory and Design (with VHDL and SystemVerilog)*. The MIT Press, 2013.

14 Jean-Eric Pin. On reversible automata. In Imre Simon, editor, *LATIN '92, 1st Latin American Symposium on Theoretical Informatics, São Paulo, Brazil, April 6-10, 1992, Proceedings*, volume 583 of *Lecture Notes in Computer Science*, pages 401–416. Springer, 1992. doi:10.1007/BFb0023844.

A Algorithm: Section 3

Algorithm 1 NP-algorithm for the Decomp problem for permutation DFAs.

```
Function isComposite($\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ : permutation DFA)
    foreach $p \in Q \setminus F$ do
        guess $U$ with $\{p\} \subseteq U \subseteq Q \setminus F$   /* guess rejecting state $U$ of some orbit-DFA, such that $U$ contains rejecting state $p$ of $\mathcal{A}$ */
        if not cover($\mathcal{A}, p, U$) then return False
    return True

Function cover($\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ : permutation DFA, $p \in Q \setminus F$, $U \subseteq Q \setminus F$)
    $\mathcal{C}_U^{\text{old}} = \varnothing$
    $\mathcal{C}_U := \{U\}$
    while $\mathcal{C}_U \neq \mathcal{C}_U^{old}$ and $|\mathcal{C}_U| < |Q|$ do
        $\mathcal{C}_U^{\text{old}} := \mathcal{C}_U$
        $\mathcal{C}_U := \mathcal{C}_U \cup \{\delta(S, \sigma) \mid S \in \mathcal{C}_U, \sigma \in \Sigma\}$
    if $|\mathcal{C}_U| \geq |Q|$ then return False   /* check that orbit-DFA is factor */
    foreach $S \in \mathcal{C}_U$ do
        if $q_I \in S$ then return True   /* check that $U$ is reachable from the inital state of the orbit-DFA */
    return False
```

B Algorithm: Section 4

Algorithm 2 Deterministic and non-deterministic version of the algorithm solving the DECOMP problem for commutative permutation DFAs.

```
Function isComposite($\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ : commutative permutation DFA)
    foreach $p \in Q \setminus F$ do
        cover_found:=False
        foreach $q \in Q \setminus F$ with $q \neq p$ do
            if cover($\mathcal{A}, p, q$) then cover_found:=True   /* covering $p$ with $w_{p,q}$ */
        if not cover_found then return False                   /* no cover found for $p$ */
    return True                                                /* all state $p$ are covered */

Function cover($\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ : commutative permutation DFA, $p, q \in Q \setminus F$)
    $s := q$
    while $s \neq p$ do                 /* eventually $s = p$ $\mathcal{A}$ is a permuation DFA */
        $s :=$ mimic($p, q, s$)                                  /* thus $s := \delta(s, w_{p,q})$ */
        if $s \in F$ then return False                      /* contradiction of covering */
    return True      /* encountered $p$ again without hitting state in $F$ */

Function mimic($\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ : commutative permutation DFA,
 $p, q, s \in Q \setminus F$)
    Assumption: $|\Sigma|$ is fixed, let $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_m\}$
    foreach $1 \leq x_1 + \cdots + x_{|\Sigma|} \leq |Q|$ do      /* possible since $|\Sigma|$ is fixed */
        if $\delta(p, \sigma_1^{x_1} \sigma_2^{x_2} \ldots \sigma_m^{x_m}) = q$ then return $\delta(s, \sigma_1^{x_1} \sigma_2^{x_2} \ldots \sigma_m^{x_m})$

Function mimic($\mathcal{A} = \langle \Sigma, Q, q_I, \delta, F \rangle$ : commutative permutation DFA,
 $p, q, s \in Q \setminus F$)
    Assumption: this algorithm is allowed to use non-determinism
    $p' := p, \ell := 0$
    while $p' \neq q$ and $\ell < |Q|$ do
        guess $\sigma \in \Sigma$                       /* iteratively contruct $w_{p,q}$ of length $\ell$ */
        $p' := \delta(p', \sigma)$, $s := \delta(s, \sigma)$, $\ell := \ell + 1$
    if $\ell = |Q|$ then return error else return $s$          /* check $q = \delta(p, w_{p,q})$ */
```

C Algorithms: Section 5

Algorithm 3 LOGSPACE-algorithm solving the BOUND-DECOMP problem for unary DFAs.

```
Function isBoundedComposite($\mathcal{A} = \langle\{a\}, Q, q_I, \delta, F\rangle$ : unary DFA, integer $k$)
    if $\mathcal{A}$ is permutation DFA then
        foreach binaryString wordCombination $\in \{0,1\}^{\log|Q|}$ with $\leq k$ ones do
            /* wordCombination represents current set in $2^{\mathcal{W}}$ */
            if testWordCombination($\mathcal{A}$, wordCombination) then return True
                    /* Set of words covering all rejecting states found */
        return False                              /* No covering set found */
    else
        call [7, Algorithm 1]
Function testWordCombination($\mathcal{A} = \langle\{a\}, Q, q_I, \delta, F\rangle$ : unary DFA,
 wordCombination : binaryString)
    foreach $q \in Q \setminus F$ do
        if not cover ($\mathcal{A}, q,$ wordCombination) then return False    /* Found state
         not covered by current set */
    return True
Function coverBySet($\mathcal{A} = \langle\{a\}, Q, q_I, \delta, F\rangle$ : unary DFA, $q \in Q \setminus F$,
 wordCombination : binaryString)
    foreach int $i$ with wordCombination[$i$] $?\!= 1$ do         /* Go through all $\leq k$
     words in the set and test if $q$ is covered */
        compute $p_1 :=$ $i$'th prime divisor of $|Q|$
        if cover($\mathcal{A}, q, \delta(q, a^{|Q|/p_i})$) then return True        /* Function cover from
         Algorithm 2 */
    return False
```

Algorithm 4 NP-algorithm solving the BOUND-DECOMP problem for commutative permutation DFAs.

```
Function isBoundedComposite(commutative permutation DFA $\mathcal{A}$, integer $k$)
    guess $\mathcal{W} := \{w_i \in \Sigma^{\leq|Q|} \mid i \leq k\}$
    foreach $p \in Q \setminus F$ do
        if not cover($\mathcal{A}, p, \mathcal{W}$) then  return False       /* Some $p$ not covered? */
    return True                                        /* all $p$ are covered */
Function cover(commutative permutation DFA $\mathcal{A}$, state $p$, set of words $\mathcal{W}$)
    foreach $w_i \in \mathcal{W}$ do
        compute $Q_{q,w_i} := \{\delta(q, w_i^{\lambda}) \mid \lambda \leq |Q|\}$
        if $Q_{q,w_i} \cap F = \varnothing$ then return True
    return False
```

Algebra and Coalgebra of Stream Products

Michele Boreale
University of Florence, Italy

Daniele Gorla
"Sapienza" University of Rome, Italy

Abstract

We study connections among polynomials, differential equations and streams over a field $\mathbb{K}$, in terms of algebra and coalgebra. We first introduce the class of (F, G)-products on streams, those where the stream derivative of a product can be expressed as a polynomial of the streams themselves and their derivatives. Our first result is that, for every (F, G)-product, there is a canonical way to construct a transition function on polynomials such that the induced unique final coalgebra morphism from polynomials into streams is the (unique) $\mathbb{K}$-algebra homomorphism – and vice-versa. This implies one can reason algebraically on streams, via their polynomial representation. We apply this result to obtain an algebraic-geometric decision algorithm for polynomial stream equivalence, for an underlying generic (F, G)-product. As an example of reasoning on streams, we focus on specific products (convolution, shuffle, Hadamard) and show how to obtain closed forms of algebraic generating functions of combinatorial sequences, as well as solutions of nonlinear ordinary differential equations.

2012 ACM Subject Classification Theory of computation → Operational semantics

Keywords and phrases Streams, coalgebras, polynomials, differential equations

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.19

Related Version *Full Version*: `https://arxiv.org/abs/2107.04455`

Supplementary Material *Software (Source Code)*: `https://local.disia.unifi.it/boreale/papers/streams.py`

1 Introduction

We investigate a connection among polynomials, differential equations and *streams*, i.e., infinite sequences of elements from a set [20]. At a very informal level, this connection can be expressed by the following correspondences: polynomials = syntax; differential equations = operational semantics; streams = abstract (denotational) semantics. There are two important motivations behind this standpoint. (1) Diverse notions of *product* (convolution, shuffle,...) arise in streams, in relation to different models – discrete computations, combinatorial sequences, analytic functions, and more [4, 20]. There is also a close analogy between several forms of products and forms of parallelism arising in concurrency. Our aim is to uniformly accommodate such diverse notions, by automatically deriving an operational semantics for polynomials that is adequate for a given *generic* stream product. (2) Once adequate polynomial syntax and operational semantics have been obtained, one can apply powerful techniques both from algebraic geometry (Gröbner bases [12]) and from coalgebra (coinduction [20]) for reasoning on streams. This includes devising algorithms for deciding stream equivalence. Again, one would like to do so in a uniform fashion w.r.t. an underlying notion of stream product.

Technically, achieving these goals amounts to defining a fully abstract semantics from polynomials to streams, which is essential for algebraic-geometric reasoning on streams. Moreover, one wants the resulting construction to be as much as possible parametric with respect to the underlying notion of stream product.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 19; pp. 19:1–19:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

As hinted above, we will pursue these goals relying on tools from algebra and coalgebra (Section 2). Indeed, it is well-known that, when polynomial coefficients and stream elements are drawn from a field $\mathbb{K}$, both polynomials and streams form $\mathbb{K}$-algebras, i.e., rings with an additional vector space structure over $\mathbb{K}$. Note that, while this algebra structure is fixed for polynomials, it varies with the underlying product for streams. On the other hand, streams also possess a *co*algebraic structure, arising from the operation of stream derivative. On the side of polynomials, it is also natural to interpret a differential equation $\dot{x}_i = p_i$ as a transition $x_i \to p_i$: thus one expects a transition structure, hence a coalgebra, over over polynomials as well. How to extend appropriately transitions from individual variables x_i to monomials and polynomials, though, depends nontrivially on the notion of stream product one wants to model.

Our first result (Section 3) is that the above outlined goals can be achieved for the class of (F, G)-products on streams, where, basically, the derivative of a product of two streams can be expressed as a polynomial of the streams themselves and their derivatives. One can then define a coalgebra structure on polynomials, depending on the given (F, G)-product and differential equations, such that the unique morphism from this coalgebra to the coalgebra of streams is also a $\mathbb{K}$-algebra *homo*morphism (and vice-versa: every homomorphism that satisfies the given differential equations is the unique morphism). Thus, full abstraction is achieved.

A major application of this result, which we view as our main contribution, is an algorithm based on an algebraic-geometric construction for deciding equivalence, i.e. if two polynomials denote the same stream (Section 4). Next, focusing on specific (F, G)-products (convolution, shuffle and Hadamard; Section 5), we show how establishing polynomial (algebraic) equalities on streams may lead to closed forms for generating functions of combinatorial sequences [13], and to solutions of nonlinear ordinary differential equations (ODEs). In the case of convolution product, we also show that the image of the coalgebra morphism is included in the set of algebraic sequences in the sense of [13].

To sum up, we make the following contributions. (1) A unifying treatment of stream products, implying that, under reasonable assumptions, coalgebra morphisms from polynomials to streams are also $\mathbb{K}$-algebra homomorphisms (full abstraction) – and viceversa. (2) An algorithm for deciding polynomial stream equivalence, that relies on the full abstraction result. (3) Based on that, methods for reasoning on generating functions and ordinary differential equations.

Due to space limitations, most proofs, as well as additional technical material, have been omitted and can be found in the full version of this paper [11].

Related work. Rutten's *stream calculus* [20, 21], a coinductive approach to the analysis of infinite sequences (streams), is a major source of inspiration for our work. [20] studies streams, automata, languages and formal power series in terms of coalgebra morphisms and bisimulation. In close analogy with classical analysis, [21] presents coinductive definitions and proofs for a calculus of *behavioural differential equations*, also called *stream differential equations* (SDEs) in later works. A number of applications to difference equations, analytical differential equations, continued fractions and problems from combinatorics, are presented. Convolution and shuffle product play a central role in the stream calculus; a duality between them, mediated by a variation of Laplace transform, exists. This duality also plays a role in our work in relation to generating functions and solutions of ODEs (Section 5). A coinductive treatment of analytic functions and Laplace transform is also presented by Escardo and Pavlovic [19]. Basold et al. [4] enrich the stream calculus with two types of products, Hadamard and infiltration, and exhibit a duality between the two, mediated by a so-called *Newton transform*. Although these works form a conceptual prerequisite of our study, they do not offer a unifying treatment of the existing disparate notions of stream product, nor any algorithmic treatment of the induced stream equivalences. Bonchi et al. [5] consider an operational approach to streams and convolution product based on weighted automata, which correspond to linear expressions. They offer an equivalence checking algorithm for such automata, and the recognized streams, based on a linear-algebraic construction; however, the polynomial case is not addressed. A related work is Bonchi et al. [3], where it is shown how linear algebra and fractions can be used to decide the equality of streams specified by linear SDEs. Here, differently from them, we can also work with polynomial SDEs.

Most closely related to the present work is Hansen, Kupke and Rutten's [14]. There the authors prove that, when the SDEs defining given operations on streams obey a GSOS syntactic format, then the final coalgebra morphism is also a homomorphism from the free term algebra to the algebra (w.r.t. the given operations) of streams [14, Sect.8]. In contrast, we work with the algebra of polynomials, which besides being a ring and vector space over $\mathbb{K}$, possesses additional structure arising from monomials. All this structure is essential for algebraic-geometric reasoning, and sets our approach apart from those based on term algebras: for one thing, in term algebras there is no obvious analog of Hilbert's basis theorem, a result deeply related to the well-ordering of monomials (cf. Dickson's lemma, [12, Ch.2]), and a crucial ingredient in our decision algorithm. One might consider more complicated GSOS frameworks enriched with equational theories, but even so we doubt one could naturally capture the relevant polynomial structure, in particular as arising from monomials. Nevertheless, a thorough exploration of these issues is an interesting direction for future research.

The GSOS format has also been discussed in the framework of *bialgebras* [14, Sect.9]. Bialgebras are a unified categorical framework that encompass both algebras, viewed as a way of modeling syntax, and coalgebras, viewed as way of describing behaviours; see [16] for a general introduction. The theory of bialgebras is very abstract in spirit, and it is not immediate to pinpoint concrete relations to our results. Furthermore, it requires a substantial background in category theory, which we have preferred to avoid here so as to keep our approach as elementary and accessible as possible. In any case, we anticipate for bialgebras similar difficulties to those discussed above for term algebras. For these reasons, we have preferred to leave the exploration of connections with bialgebras for future work.

Somewhat related to ours is the work of Winters on coalgebra and polynomial systems: see e.g. [23, Ch.3]. Importantly, Winter considers polynomials in *noncommuting* variables: under suitable assumptions, this makes his systems of equations isomorphic to certain context-free grammars; see also [17]. The use of noncommuting variables sets Winter's treatment in a totally different mathematical realm, where the algebraic geometric concepts we rely on here, like ideals and Gröbner bases, are not applicable.

We also mention [7, 10], that adopt a coinductive approach to reason on polynomial ODEs. The ring of multivariate polynomials is employed as a syntax, with *Lie derivatives* inducing a transition structure. An algebraic-geometric algorithm to decide polynomial equivalence is presented. This algorithm as well has inspired our decision method: in particular, as Lie derivatives are precisely the transition structure induced in our framework by the shuffle product, the decision algorithms of [7, 10] are in essence a special case of our algorithm in Section 4. Furthermore, [8, 9] extend the framework of [7, 10] to polynomial partial differential equations, which pose significative additional challenges.

Relations with work in enumerative combinatorics [13, 22] are discussed in Section 5.

2 Background

2.1 Polynomials and differential equations

Let us fix a finite, non empty set of symbols or *variables* $X = \{x_1, \ldots, x_n\}$ and a distinct variable $x \notin X$. Informally, x will act as the independent variable, while $x_1, ..., x_n$ will act as dependent variables, or functions, defined by differential equations (see below). We fix a generic field $\mathbb{K}$ of characteristic 0; $\mathbb{K} = \mathbb{R}$ and $\mathbb{K} = \mathbb{C}$ are typical choices. We let $\mathcal{P} := \mathbb{K}[x, x_1, ..., x_n]$, ranged over by $p, q, ...$, be the set of polynomials with coefficients in $\mathbb{K}$ and indeterminates in $\{x\} \cup X$. We let $\mathcal{M}$, ranged over by $m, m', ...$, be the set of *monomials*, that is the free commutative monoid generated by $\{x\} \cup X$. As usual, we shall denote polynomials as formal finite sums of distinct monomials with nonzero coefficients in $\mathbb{K}$: $p = \sum_{i \in I} r_i m_i$, for $r_i \in \mathbb{K}$. By slight abuse of notation, we shall write the zero polynomial and the empty monomial as 0 and 1, respectively. Over $\mathcal{P}$, one can define the usual operations of sum $p + q$ and product $p \cdot q$, with 0 and 1 as identities, and enjoying commutativity, associativity and distributivity, which make $\mathcal{P}$ a ring; multiplication of $p \in \mathcal{P}$ by a scalar $r \in \mathbb{K}$, denoted rp, is also defined and makes $(\mathcal{P}, +, 0)$ a vector space over $\mathbb{K}$. Therefore, $(\mathcal{P}, +, \times, 0, 1)$ forms a $\mathbb{K}$-algebra.

We shall also fix a set $\mathcal{D} = \{\dot{x}_1 = p_1, ..., \dot{x}_n = p_n\}$ of *differential equations*, one for each $x_i \in X$, with $p_i \in \mathcal{P}$. An *initial condition* for $\mathcal{D}$ is a vector $\rho = (r_1, ..., r_n) \in \mathbb{K}^n$. The pair $(\mathcal{D}, \rho)$ forms an *initial value problem*. The vectors p_i on the right-hand side of the equations are called *drifts*, and $F = (p_1, ..., p_n)$ is a *vector field*. Informally, each $x_i \in X$ represents a placeholder for a function whose derivative is given by p_i, and whose value at the origin is $x_i(0) = r_i$. This terminology is borrowed from the theory of differential equations. Note, however, that depending on the semantics of polynomial product one adopts (see next section), $\mathcal{D}$ can be given diverse interpretations, including *stream* differential equations (SDE, for convolution, see next subsection) in the sense of Rutten [20], and of course *ordinary* differential equations (ODEs, for shuffle).

Notationally, it will be sometimes convenient to regard $\mathcal{D}$ and ρ as functions $\mathcal{D} : X \to \mathcal{P}$ and $\rho : X \to \mathbb{K}$, respectively, such that $\mathcal{D}(x_i) = p_i$ and $\rho(x_i) = r_i$. It is also convenient to extend $\mathcal{D}$ and ρ to x by letting $\mathcal{D}(x) = 1$ and $\rho(x) = 0$; note that, seen as an initial value problem, the last two equations define the identity function. Finally, we let x_0 denote x and, when using $\mathcal{D}$ and ρ as functions, use x_i as a metavariable on $\{x\} \cup X$: this makes $\mathcal{D}(x_i)$ and $\rho(x_i)$ well defined for $0 \leq i \leq n$.

2.2 Streams

We let $\Sigma\langle\mathbb{K}\rangle := \mathbb{K}^\omega$, ranged over by $\sigma, \tau, ...$, denote the set of *streams*, that is infinite sequences of elements from $\mathbb{K}$: $\sigma = (r_0, r_1, r_2, ...)$ with $r_i \in \mathbb{K}$. Often $\mathbb{K}$ is understood from the context and we shall simply write Σ rather than $\Sigma\langle\mathbb{K}\rangle$. When convenient, we shall explicitly consider a stream σ as a function from $\mathbb{N}$ to $\mathbb{K}$ and, e.g., write $\sigma(i)$ to denote the i-th element of σ. By slightly overloading the notation, and when the context is sufficient to disambiguate, the stream $(r, 0, 0, ...)$ $(r \in \mathbb{K})$ will be simply denoted by r, while the stream $(0, 1, 0, 0, ...)$ will be denoted by x; see [20] for motivations behind these notations[1]. Furthermore, a stream made up of all the same element $r \in \mathbb{K}$ will be denoted as $\underline{r} = (r, r, ...)$. One defines the *sum* of two streams σ and τ as the stream $\sigma + \tau$ defined by: $(\sigma + \tau)(i) := \sigma(i) + \tau(i)$ for each $i \geq 0$, where the $+$ on the right-hand side denotes the sum in $\mathbb{K}$. Sum enjoys the usual commutativity

[1] In particular, overloading of the symbol x is motivated by the fact that our semantics of polynomials maps the variable x to the stream $(0, 1, 0, 0, ...)$.

and associativity properties, and has the stream $0 = (0, 0, \ldots)$ as an identity. Various forms of stream products can also be considered – this is indeed a central theme of our paper. In particular, the *convolution product* $\sigma \times \tau$ and the *shuffle product* $\sigma \otimes \tau$ are defined as follows: $(\sigma \times \tau)(i) := \sum_{0 \le j \le i} \sigma(j) \cdot \tau(i - j)$ and $(\sigma \otimes \tau)(i) := \sum_{0 \le j \le i} \binom{i}{j} \sigma(j) \cdot \tau(i - j)$, where operations on the right-hand side are carried out in $\mathbb{K}$ and $i \ge 0$. The above operations enjoy alternative, easier to handle formulations based on stream differential equations – see next subsection; there, a crucial notion will be the *derivative* of a stream σ, that is the stream σ' obtained from σ by removing its first element.

Both products are commutative, associative, have $1 = (1, 0, 0, \ldots)$ as an identity, and distribute over $+$; multiplication of $\sigma = (r_0, r_1, \ldots)$ by a scalar $r \in \mathbb{K}$, denoted $r\sigma = (r\, r_0, r\, r_1, \ldots)$, is also defined and makes $(\Sigma, +, 0)$ a vector space over $\mathbb{K}$. Therefore, $(\Sigma, +, \pi, 0, 1)$ forms a $\mathbb{K}$-algebra for each of the considered product operations π. Let us record the following useful properties for future use: $x \times \sigma = (0, r_0, r_1, \ldots)$ and $r\, \pi\, \sigma = (r\, r_0, r\, r_1, \ldots)$, where $r \in \mathbb{K}$ and $\pi \in \{\times, \otimes\}$. In view of the second equation above, $r\, \pi\, \sigma$ coincides with $r\sigma$. The first equation above leads to the so called fundamental theorem of the stream calculus, whereby for each $\sigma \in \Sigma$

$$\sigma = \sigma(0) + x \times \sigma' \,. \tag{1}$$

Less commonly found forms of products, like Hadamard and Infiltration products, will be introduced in the next subsection; equations similar to (1) exist also for such products [4, 14].

2.3 Coalgebras and bisimulation

We quickly review some basic definitions and results about coalgebras and bisimulation; see e.g. [20] for a comprehensive treatment. A *(stream) coalgebra with outputs in* $\mathbb{K}$ is an automaton $C = (S, \delta, o)$, where S is a nonempty set of *states*, $\delta : S \to S$ is the *transition* function, and $o : S \to \mathbb{K}$ is the *output* function. A *bisimulation* on C is a binary relation $R \subseteq S \times S$ such that, whenever $(s, t) \in R$, then $o(s) = o(t)$ and $(\delta(s), \delta(t)) \in R$. As usual, there always exists a largest bisimulation on C, denoted $\sim$; it is the union of all bisimulations and it is an equivalence relation on S. Given two coalgebras C_1 and C_2, a *coalgebra morphism* between them is a function $\mu : S_1 \to S_2$ from the states of C_1 to the states of C_2 that preserves transitions and outputs, that is (with obvious notation): $\mu(\delta_1(s)) = \delta_2(\mu(s))$ and $o_1(s) = o_2(\mu(s))$, for each $s \in S_1$. Coalgebra morphisms preserve bisimilarity, in the sense that $s \sim_1 t$ in C_1 if and only if $\mu(s) \sim_2 \mu(t)$ in C_2. A coalgebra C_0 is *final* in the class of coalgebras with outputs in $\mathbb{K}$ if, from every coalgebra C in this class, there exists a unique morphism μ from C_0 to C. In this case, $\sim_0$ in C_0 coincides with equality, and the following *coinduction principle* holds: for every C and $s \sim t$ in C, it holds that $\mu(s) = \mu(t)$ in C_0.

The set of streams Σ can be naturally given a stream coalgebra structure $(\Sigma, (\cdot)', o(\cdot))$, as follows. The *output* of a stream $\sigma = (r_0, r_1, \ldots)$ is $o(\sigma) := r_0$ and its *derivative* is $\sigma' := (r_1, r_2, \ldots)$, that is σ' is obtained from σ by removing its first element, that constitutes the output of σ. In fact, this makes Σ final in the class of all coalgebras with outputs in $\mathbb{K}$ [20]. This also implies that one can prove equality of two streams by exhibiting an appropriate bisimulation relation relating them (coinduction).

It is sometimes convenient to consider an enhanced form of bisimulation on Σ that relies on the notion of *linear closure*.[2] Given a relation $R \subseteq \Sigma \times \Sigma$, its linear closure $\widehat{R}$ is the set of pairs of the form $(\sum_{i=1}^{n} r_i \sigma_i \ , \ \sum_{i=1}^{n} r_i \tau_i)$, where $n \in \mathbb{N}$, $(\sigma_i, \tau_i) \in R$ and $r_i \in \mathbb{K}$, for every

[2] More general notions that we could have used here are *contextual closure* (see [4, Thm. 2.4] and works on distributive laws for bialgebras [6]. However, the simpler notion of linear closure suffices for our purposes here.

$i \in \{1, \ldots, n\}$. We say that R is a *bisimulation up to linearity* if, for every $(\sigma, \tau) \in R$, it holds that $o(\sigma) = o(\tau)$ and $(\sigma', \tau') \in \widehat{R}$. If R is a bisimulation up to linearity, then $\widehat{R}$ is a bisimulation [20]; since by definition $R \subseteq \widehat{R}$, this implies that $R \subseteq \sim$, the bisimilarity on streams, which coincides with equality.

A *stream differential equation* (SDE) in the unknown σ is a pair of equations of the form $\sigma(0) = r$ and $\sigma' = \phi$, for $r \in \mathbb{K}$ and a stream expression ϕ (that can depend on σ or its components, or even on σ' itself). Under certain conditions on ϕ [14, 20], it can be proven that there is a unique stream σ satisfying the above SDE. In this paper, we shall focus on the case where ϕ is represented by a polynomial expression – this will be formalized in the next section. For the time being, we observe that the product operations defined in the preceding subsection enjoy a formulation in terms of SDEs. In particular (see [4, 14, 20]), for given σ and τ, their convolution and shuffle products are the unique streams satisfying the following SDEs (recall that, as a stream, x denotes $(0, 1, 0, 0, \ldots)$):

$$(\sigma \times \tau)(0) = \sigma(0) \cdot \tau(0) \qquad (\sigma \times \tau)' = \sigma' \times \tau + \sigma \times \tau' - x \times \sigma' \times \tau' \tag{2}$$

$$(\sigma \otimes \tau)(0) = \sigma(0) \cdot \tau(0) \qquad (\sigma \otimes \tau)' = \sigma' \otimes \tau + \sigma \otimes \tau' \,. \tag{3}$$

From the last equation, note the analogy between shuffle and interleaving of languages. Moreover, the derivative of convolution product is usually defined as $(\sigma \times \tau)' = \sigma' \times \tau + \sigma(0) \times \tau'$; however, we prefer the formulation in (2) because it is symmetric. Two additional examples of stream products are introduced below; see [4] for the underlying motivations. The *Hadamard product* $\odot$ and the *infiltration product* $\uparrow$ can be defined by the following two SDEs.

$$(\sigma \odot \tau)(0) = \sigma(0)\tau(0) \qquad (\sigma \odot \tau)' = \sigma' \odot \tau' \tag{4}$$

$$(\sigma \uparrow \tau)(0) = \sigma(0)\tau(0) \qquad (\sigma \uparrow \tau)' = (\sigma' \uparrow \tau) + (\sigma \uparrow \tau') + (\sigma' \uparrow \tau') \,. \tag{5}$$

Hadamard product $\odot$ is reminiscent of synchronization in concurrency theory and has $\underline{1} := (1, 1, 1, \ldots)$ as an identity; it is just the componentwise product of two streams, i.e. $(\sigma \odot \tau)(i) = \sigma(i)\tau(i)$, for every $i \geq 0$. Infiltration product $\uparrow$ is again reminiscent of a notion in concurrency theory, namely the fully synchronized interleaving; it has $1 = (1, 0, 0, \ldots)$ as an identity.

3 (Co)algebraic semantics of polynomials and differential equations

The main result of this section is that, once fixed an initial value problem $(\mathcal{D}, \rho)$, for every product π (with identity 1_π) defined on streams and satisfying certain syntactic conditions, one can build a coalgebra over polynomials such that the corresponding final *morphism* into Σ is also a $\mathbb{K}$-algebra *homomorphism* from $(\mathcal{P}, +, \times, 0, 1)$ to $(\Sigma, +, \pi, 0, 1_\pi)$. In essence, the polynomial syntax and operational semantics reflects exactly the algebraic and coalgebraic properties of the considered π on streams.

To make polynomials a coalgebra, we need to define the output $o : \mathcal{P} \to \mathbb{K}$ and transition $\delta : \mathcal{P} \to \mathcal{P}$ functions. The definition of $o(\cdot)$ is straightforward and only depends on the given initial conditions ρ: we let $o := o_\rho$ be the homomorphic extension of ρ, seen as a function defined over $\{x\} \cup X$, to $\mathcal{P}$. Equivalently, seeing ρ as a point in $\mathbb{K}^{n+1}$, we let $o_\rho(p) := p(\rho)$, that this the polynomial p evaluated at the point ρ. It can be easily checked that $o_\rho(1) = 1$.

The definition of δ, on the other hand, depends on π and is not straightforward. We will confine to products π satisfying SDEs of the form: $(\sigma \,\pi\, \tau)' = F(\sigma, \tau, \ldots)$, for a given *polynomial* function F. Then we will require that δ on polynomials mimics this equation. For instance, in the case of shuffle product, we expect that $\delta(pq) = p\delta(p) + q\delta(p)$. Therefore, our first step is to precisely define the class of products on streams that satisfy a polynomial SDE. To this purpose, in what follows we shall consider polynomials $G(y_1) \in \mathbb{K}[y_1]$ and $F(x, y_1, ..., y_4) \in \mathbb{K}[x, y_1, y_2, y_3, y_4]$. These can be identified with polynomial functions on streams: we shall write $G(\sigma_1)$, $F(x, \sigma_1, ..., \sigma_4)$ for the evaluation of G, F in $(\Sigma, +, \pi, 0, 1_\pi)$ with specific streams $x = (0, 1, 0, ...)$ and $\sigma_1, ..., \sigma_4$.

▶ **Definition 3.1** ((F, G)-product on streams). *Let $(\Sigma, +, \pi, 0, 1_\pi)$ be a $\mathbb{K}$-algebra, $F \in \mathbb{K}[x, y_1, y_2, y_3, y_4]$ and $G \in \mathbb{K}[y_1]$. We say that π is a* (F, G)-product *if, for each $\sigma, \tau \in \Sigma$, the following equations are satisfied:*

1. $(\sigma\ \pi\ \tau)(0) = \sigma(0)\tau(0)$;
2. $(\sigma\ \pi\ \tau)' = F(x, \sigma, \sigma', \tau, \tau')$;
3. $1_\pi(0) = 1$ *and* $1'_\pi = G(1_\pi)$.

▶ Remark 3.2. Notice that $1_\pi(0) = 1$ in Definition 3.1(3) is a necessary condition, that follows from Definition 3.1(1). Indeed, let $1_\pi(0) = r \in \mathbb{K}$. Since 1_π is the identity of π, for every σ we must have $\sigma\, \pi\, 1_\pi = \sigma$, hence $(\sigma\, \pi\, 1_\pi)(0) = \sigma(0)$. On the other hand, by Definition 3.1(1), $(\sigma\, \pi\, 1_\pi)(0) = \sigma(0)\, 1_\pi(0) = \sigma(0)\, r$. As σ is arbitrary, we can take $\sigma(0) \neq 0$ and multiply $\sigma(0)\, r = \sigma(0)$ by $\sigma(0)^{-1}$; this gives $r = 1$. However, we prefer to keep $1_\pi(0) = 1$ explicit in the definition, for the sake of clarity. Finally, let us note that the general theory of SDEs [14] ensures that conditions (1), (2), (3) in Definition 3.1 univocally define a binary operation π on streams, but in general not that π enjoys the ring axioms for product, a fact that we must assume from the outset.

▶ **Example 3.3.** For the products introduced in Section 2, the pairs of polynomials (F, G) are as defined as follows.

- $F_\times = y_2y_3 + y_1y_4 - xy_2y_4$. Note that $F_\times = y_2y_3 + (y_1 - xy_2)y_4$, where $y_1 - xy_2$ corresponds to $\sigma - x \times \sigma' = \sigma(0)$; this gives the asymmetric definition of convolution.
- $F_\otimes = y_2y_3 + y_1y_4$.
- $F_\odot = y_2y_4$.
- $F_\uparrow = y_2y_3 + y_1y_4 + y_2y_4$.

The identity stream for convolution, shuffle and infiltration is defined by $1_\pi(0) = 1$ and $1'_\pi = 0$, i.e., in these cases the polynomial G is 0. For the Hadamard product, the identity is given by $1_\pi(0) = 1$ and $1'_\pi = 1_\pi$, i.e., the polynomial G in this case is y_1.

Given a (F, G)-product π on streams, δ_π is defined in a straightforward manner on monomials, then extended to polynomials by linearity. Below, we assume a total order on variables $x_0 < x_1 < \cdots < x_n$ and, for $m \neq 1$, let $\min(m)$ denote the smallest variable occurring in m w.r.t. such a total order[3].

▶ **Definition 3.4** (transition function δ_π). *Let π be a (F, G)-product on streams. We define $\delta_\pi : \mathcal{P} \to \mathcal{P}$ by induction on the size of $p \in \mathcal{P}$ as follows.*

$$\delta_\pi(1) = G(1) \tag{6}$$

$$\delta_\pi(x_i) = \mathcal{D}(x_i) \tag{7}$$

$$\delta_\pi(x_i\, m) = F(x, x_i, \delta_\pi(x_i), m, \delta_\pi(m)) \quad (m \neq 1,\ \ x_i = \min(x_i m)) \tag{8}$$

$$\delta_\pi\left(\sum_{i \in I} r_i\, m_i\right) = \sum_{i \in I} r_i\, \delta_\pi(m_i)\,. \tag{9}$$

[3] In Definition 3.4, we are in effect totally ordering monomials by graded lexicographic order (grlex, see [12, Ch.1]), and then proceeding by induction on this order.

Returning to the products defined in Section 2, we have:

$$\delta_\pi(1) \quad = \quad \begin{cases} 0 & \text{for } \pi \in \{\times, \otimes, \uparrow\} \text{ (convolution, shuffle, infiltration)} \\ 1 & \text{for } \pi = \odot \text{ (Hadamard)} \end{cases}$$

$$\delta_\pi(x_i\, m) \quad = \quad \begin{cases} \mathcal{D}(x_i)\cdot m + x_i \cdot \delta_\pi(m) - x\cdot \mathcal{D}(x_i)\cdot \delta_\pi(m) & \text{for } \pi = \times \text{ (convolution)} \\ \mathcal{D}(x_i)\cdot m + x_i \cdot \delta_\pi(m) & \text{for } \pi = \otimes \text{ (shuffle)} \\ \mathcal{D}(x_i)\cdot \delta_\pi(m) & \text{for } \pi = \odot \text{ (Hadamard)} \\ \mathcal{D}(x_i)\cdot m + \mathcal{D}(x_i)\cdot \delta_\pi(m) + x_i \cdot \delta_\pi(m) & \text{for } \pi = \uparrow \text{ (infiltration)}\,. \end{cases}$$

We must now impose certain additional sanity conditions on F, to ensure that the final coalgebra morphism induced by δ_π, as just defined, is also an algebra homomorphism. In the rest of the paper, we will make use of the following abbreviation $F_\pi[p;q] := F(x, p, \delta_\pi(p), q, \delta_\pi(q))$. The necessity of the following conditions is self-evident, if one thinks of $F_\pi[p;q]$ as $\delta_\pi(p\cdot q)$ (see Lemma 3.6 below).

▶ **Definition 3.5** (well-behaved F). *Let π be a (F,G)-product on streams. We say that π is* well-behaved *if the following equalities hold, for every $p,q \in \mathcal{P}$, $m_1, m_2, m_i \in \mathcal{M}$, $x_i \in \{x\} \cup X$ and $r_i \in \mathbb{K}$:*

$$F_\pi[1;q] = \delta_\pi(q) \tag{10}$$

$$F_\pi[x_i m_1; m_2] = F_\pi[m_1; x_i m_2] \tag{11}$$

$$F_\pi\left[\sum_{i\in I} r_i\, m_i \ ;\ q\right] = \sum_{i\in I} r_i\, F_\pi[m_i;q] \tag{12}$$

$$F_\pi[p;q] = F_\pi[q;p]\,. \tag{13}$$

All products defined in Section 2 are well-behaved. The following key technical result connects morphism to homomorphism properties induced by π and is crucial in the proof of Theorem 3.7, that is the main result of this section.

▶ **Lemma 3.6.** *Let π be a well-behaved (F,G)-product. Then, for every $p,q \in \mathcal{P}$, it holds that $\delta_\pi(p\cdot q) = F_\pi[p;q]$.*

▶ **Theorem 3.7.** *Let π be a well-behaved (F,G)-product. Then the (unique) coalgebra morphism μ_π from $(\mathcal{P}, \delta_\pi, o_\rho)$ to $(\Sigma, (\cdot)', o)$ is a $\mathbb{K}$-algebra homomorphism from $(\mathcal{P}, +, \cdot, 0, 1)$ to $(\Sigma, +, \pi, 0, 1_\pi)$.*

Intuitively, the proof consists in showing that μ_π preserves all the operations in $\mathcal{P}$, by exhibiting in each case an appropriate bisimulation relation in $\Sigma\times\Sigma$ and then applying coinduction. The most crucial case is product, where one shows that the relation consisting of all pairs $(\mu_\pi(p_1\cdot\ldots\cdot p_k)\,,\, \mu_\pi(p_1)\,\pi\ldots\pi\,\mu_\pi(p_k))$ $(k>0)$ is a bisimulation up to linearity. Lemma 3.6 is used to prove that μ_π preserves transitions: e.g., by letting $p = p_2\cdot\ldots\cdot p_k$, it allows one to conclude that the pair of derivatives $\mu_\pi(p_1\cdot p)' = \mu_\pi(F_\pi[p_1;p])$ and (slightly abusing the $F_\pi[\cdot;\cdot]$ notation) $(\mu_\pi(p_1)\,\pi\,\mu_\pi(p))' = F_\pi[\mu_\pi(p_1);\mu_\pi(p)]$ are still in relation, up to linearity.

To conclude the section, we also present a sort of converse of the previous theorem. That is, μ_π is the only $\mathbb{K}$-algebra homomorphism that respects the initial value problem, i.e. that satisfies $\mu_\pi(x_i)' = \mu_\pi(\mathcal{D}(x_i))$ and $\mu_\pi(x_i)(0) = \rho(x_i)$. This is an immediate corollary of the following result and of the uniqueness of the final coalgebra morphism.

▶ **Proposition 3.8.** *Let π be a well-behaved (F,G)-product and ν be a $\mathbb{K}$-algebra homomorphism from $(\mathcal{P}, +, \cdot, 0, 1)$ to $(\Sigma, +, \pi, 0, 1_\pi)$ that respects the initial value problem $(\mathcal{D}, \rho)$. Then, ν is a coalgebra morphism from $(\mathcal{P}, \delta_\pi, o_\rho)$ to $(\Sigma, (\cdot)', o)$.*

4 Deciding stream equality

One benefit of a polynomial syntax is the possibility of applying techniques from algebraic geometry to reason about stream equality. We will devise an algorithm for checking whether two given polynomials are semantically equivalent, that is, are mapped to the same stream under μ_π. Note that, by linearity of $\mu_\pi(\cdot)$, we have that $\mu_\pi(p) = \mu_\pi(q)$ if and only if $\mu_\pi(p) - \mu_\pi(q) = \mu_\pi(p - q) = 0$. Therefore, checking semantic equivalence of two polynomials reduces to the problem of checking if a polynomial is equivalent (bisimilar) to 0. Before introducing the actual algorithm for checking this, we quickly recall a few notions from algebraic geometry; see [12, Ch.1–4] for a comprehensive treatment.

A set of polynomials $I \subseteq \mathcal{P}$ is an *ideal* if $0 \in I$ and, for all $p_1, p_2 \in I$ and $q \in \mathcal{P}$, it holds that $p_1 + p_2 \in I$ and $q \cdot p_1 \in I$. Given a set of polynomials S, the *ideal generated by* S is

$$\langle\, S \,\rangle := \left\{ \sum_{j=1}^{k} q_j \cdot p_j \;:\; k \geq 0 \;\wedge\; \forall j \leq k.(q_j \in \mathcal{P} \wedge p_j \in S) \right\} .$$

By the previous definition, we have that $\langle\, \emptyset \,\rangle := \{0\}$. Trivially, $I = \langle\, S \,\rangle$ is the smallest ideal containing S, and S is called a set of *generators* for I. It is well-known that every ideal I admits a finite set S of generators (Hilbert's basis theorem). By virtue of this result, any infinite ascending chain of ideals, $I_0 \subseteq I_1 \subseteq I_2 \subseteq \cdots \subseteq \mathcal{P}$, stabilizes in a finite number of steps: that is, there is $k \geq 0$ s.t. $I_{k+j} = I_k$ for each $j \geq 0$ (Ascending Chain Condition, ACC). A key result due to Buchberger is that, given a finite $S \subseteq \mathcal{P}$, it is possible to decide whether $p \in I = \langle\, S \,\rangle$, for any polynomial p. As a consequence, also ideal inclusion $I_1 \subseteq I_2$ is decidable, given finite sets of generators for I_1, I_2. These facts are consequences of the existence of a set of generators B for I, called *Gröbner basis*, with a special property: $p \in I$ if and only if $p \bmod B = 0$, where "mod B" denotes the remainder of the multivariate polynomial division of p by B. There exist algorithms to build Gröbner bases which, despite their exponential worst-case complexity, turn out to be effective in many practical cases [12, Ch.4].

In what follows, we fix a well-behaved (F, G)-product π, and let δ_π and μ_π denote the associated transition function and coalgebra morphism. Moreover, we denote by $p^{(j)}$ the j-th derivative of p, i.e. $p^{(0)} := p$ and $p^{(j+1)} := \delta_\pi(p^{(j)})$. The actual decision procedure is presented below as Algorithm 1. Intuitively, to prove that $\mu_\pi(p) = 0$, one might check if $o_\rho(p^{(j)}) = 0$ for every j, which is of course non effective. But due to ACC, at some point $p^{(j)} \in \langle\, \{p^{(0)}, \ldots, p^{(j-1)}\} \,\rangle$, which implies the condition $o_\rho(p^{(j)}) = 0$ holds for *all* j's. The correctness of this algorithm can be proven easily, under an additional mild condition on F: we require that $F \in \langle\, \{y_3, y_4\} \,\rangle$ seen as an ideal in $\mathbb{K}[x, y_1, ..., y_4]$. Explicitly, $F = h_1 y_3 + h_2 y_4$ for some $h_1, h_2 \in \mathbb{K}[x, y_1, ..., y_4]$. The polynomials F for the products in Section 2 all satisfy this condition: for example, $F_\times = y_2 y_3 + (y_1 - x y_2) y_4$.

Algorithm 1 Checking equivalence to zero.

```
Input: $p \in \mathcal{P}$, a well-behaved $(F, G)$-product $\pi$
Output: YES ($\mu_\pi(p) = 0$) or NO ($\mu_\pi(p) \neq 0$)
for all $k \geq 0$ do
    if $o_\rho(p^{(k)}) \neq 0$ then return NO
    if $p^{(k)} \in \langle\, \{p^{(0)}, \ldots, p^{(k-1)}\} \,\rangle$ then return YES
end for
```

▶ **Theorem 4.1.** *Let π be a well-behaved (F, G)-product, with $F \in \langle\, \{y_3, y_4\} \,\rangle$. Algorithm 1 terminates, and returns YES if and only if $\mu_\pi(p) = 0$.*

Proof. Non termination for some input polynomial p would imply that, for all $k \geq 0$, $p^{(k+1)} \notin I_k := \langle \{p^{(0)}, \ldots, p^{(k)}\} \rangle$. This in turn would imply an ever ascending chain of ideals $I_0 \subsetneq I_1 \subsetneq \cdots$, contradicting ACC.

If the algorithm returns NO, then for some k we must have (recall that $\sigma^{(k)}$ stands for the k-th stream derivative of σ): $o_\rho(p^{(k)}) = o(\mu_\pi(p)^{(k)}) = (\mu_\pi(p)^{(k)})(0) \neq 0$, thus $\mu_\pi(p) \neq 0$.

Assume now the algorithm returns YES. Then there exists $k \geq 0$ such that $o_\rho(p^{(j)}) = 0$, for every $0 \leq j \leq k$, and $p^{(k)} \in \langle \{p^{(0)}, \ldots, p^{(k-1)}\} \rangle$. Excluding the trivial case $p = 0$, we can assume $k \geq 1$. If we prove that $p^{(k+j)} \in \langle \{p^{(0)}, \ldots, p^{(k-1)}\} \rangle$ for every $j \geq 0$, the thesis follows: indeed, by $p^{(k+j)} = \sum_{i=0}^{k-1} q_i \cdot p^{(i)}$, for some $q_i \in \mathcal{P}$, and by $o_\rho(p^{(i)}) = 0$ for every $0 \leq i \leq k-1$, it also follows $(\mu_\pi(p))(j) = (\mu_\pi(p))^{(j)}(0) = o_\rho(p^{(k+j)}) = 0$. Now the proof that $p^{(k+j)} \in \langle \{p^{(0)}, \ldots, p^{(k-1)}\} \rangle$ is by induction on j. The base case ($j = 0$) holds by assumption. For the induction step, let us consider $p^{(k+j+1)}$. By definition, $p^{(k+j+1)} = \delta_\pi(p^{(k+j)})$; by induction $p^{(k+j)} = \sum_{i=0}^{k-1} q_i \cdot p^{(i)}$, for some $q_i \in \mathcal{P}$. By (9) and Lemma 3.6, $p^{(k+j+1)} = \sum_{i=0}^{k-1} \delta_\pi(q_i \cdot p^{(i)}) = \sum_{i=0}^{k-1} F_\pi[q_i; p^{(i)}]$. By hypothesis $F \in \langle \{y_3, y_4\} \rangle$, hence $F_\pi[q_i; p^{(i)}] \in \langle \{p^{(i)}, p^{(i+1)}\} \rangle$, for every i, therefore $F_\pi[q_i; p^{(i)}] \in \langle \{p^{(0)}, \ldots, p^{(k-1)}\} \rangle$, as by hypothesis $p^{(k)} \in \langle \{p^{(0)}, \ldots, p^{(k-1)}\} \rangle$. This suffices to conclude. ◀

We first illustrate the algorithm with a simple, linear example.

▶ **Example 4.2** (Fibonacci numbers)**.** Consider the initial value problem $(\mathcal{D}, \rho)$ given by the following equations.

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = x_1 + x_2 \end{cases} \qquad \begin{cases} \rho(x_1) = 0 \\ \rho(x_2) = 1\,. \end{cases} \tag{14}$$

Let us consider here the convolution product ×. It is easily checked that x_1 defines the Fibonacci numbers: $\mu_\times(x_1) = (0, 1, 1, 2, 3, 5, 8, 13, \ldots)$. We want to prove the following equation:

$$\mu_\times(x_1 \cdot (1 - x - x^2)) = \mu_\times(x)\,. \tag{15}$$

Equivalently, using Algorithm 1, we check that $\mu_\times(x_1 \cdot (1 - x - x^2) - x) = 0$. Let $p(x, x_1) := x_1 \cdot (1 - x - x^2) - x$. Then, an execution of Algorithm 1 consists of the following steps.

- **(k = 0):** $\rho(p) = p(0, 1) = 0$ and $p^{(0)} = p(x, x_1) \notin \langle \emptyset \rangle = \{0\}$.
- **(k = 1):** $p^{(1)} = x_2 \cdot (1 - x - x^2) - x_1 \cdot (1 + x) - x \cdot x_2 \cdot (1 + x) - 1 = x_2 - x_1 - x_1 x - 1$. Hence, $\rho(p^{(1)}) = 1 - 1 = 0$ and $p^{(1)} \notin \langle p \rangle$.
- **(k = 2):** $p^{(2)} = x_1 + x_2 - x_2 - (x_2 x + x_1 - x x_2) = 0$. Hence, $\rho(p^{(2)}) = 0$ and trivially $p^{(2)} \in \langle p, p^{(1)} \rangle$.

We conclude that $\mu_\times(p) = 0$.

We now discuss a nonlinear example based on shuffle product.

▶ **Example 4.3** (double factorial of odd numbers)**.** Consider the initial value problem $(\mathcal{D}, \rho)$ given by the following equation.

$$\dot{y} = y^3 \qquad \rho(y) = 1\,. \tag{16}$$

Let us consider here the shuffle product $\otimes$. It is easily checked that $\mu_\otimes(y) = (1, 1, 3, 15, 105, 945, 10395, 135135, \ldots)$, the sequence of double factorials of odd numbers (sequence A001147 in [1]). We want to check the following equation

$$\mu_\otimes(y^2(x - 1/2) + 1/2) = 0\,. \tag{17}$$

using Algorithm 1. Let $q(x, y) := y^2(x - 1/2) + 1/2$. An execution of Algorithm 1 consists of the following steps.

- **$(k = 0)$:** $\rho(q) = q(0,1) = 0$ and $q^{(0)} = q(x,y) \notin \langle \emptyset \rangle = \{0\}$.
- **$(k = 1)$:** $q^{(1)} = 2y^4x - y^4 + y^2 = 2y^2q$, hence $q^{(1)} \in \langle q \rangle$.

We conclude that $\mu_\otimes(q) = 0$.

▶ Remark 4.4. Note that we can define the generating function associated to Fibonacci numbers, that is the function $g(z)$ whose Taylor series expansion is $\sum_{j\geq 0} f_j z^j$ (where f_j are the Fibonacci numbers); such a generating function is

$$g(z) = \frac{z}{1 - z - z^2}\,. \tag{18}$$

Now, from [4] it is known that the convolution product admits an inverse of a given stream σ whenever $\sigma(0) \neq 0$. Thus, from (15) we obtain $\mu_\times(x_1) = \mu_\times(x) \times (\mu_\times(1 - x - x^2))^{-1} = \frac{\mu_\times(x)}{1-\mu_\times(x)-\mu_\times(x)^2}$, where we use the usual notation $\frac{\sigma}{\tau}$ to denote $\sigma \times \tau^{-1}$. This equation for $\mu_\times(x_1)$ is structurally identical to (18): this is of course no coincidence, as algebraic identities on streams correspond exactly to algebraic identities on generating functions. This will be made precise in the next section – see in particular Proposition 5.2.

Similarly, the equivalence $\mu_\otimes(p) = 0$ obtained for the double factorial equations, when solved algebraically for x_1 yields the exponential generating function for A001147, that is $g(z) = \sqrt{1/(1-2z)}$: see Example 5.8 in Subsection 5.3.

We finally point out that Algorithm 1 can be easily modified to actually *find* all polynomials p, up to a prescribed degree, s.t. $\mu_\pi(p) = 0$, along the lines of a similar procedure in [10]. Indeed, we actually found the polynomials in both examples above using this modified algorithm[4].

5 Shuffle, convolution and generating functions

We study the relation of the shuffle and convolution products, and of the corresponding morphisms, with algebraic sequences arising in enumerative combinatorics [13, 22], and with solutions of ordinary differential equations; Hadamard product plays also a role in connecting the other two products. Our aim here is not to prove any new identity, but rather to relate our framework with certain well established notions and results in these fields. In particular, we will argue that our results can be useful for combinatorial reasoning on sequences and ODEs: this means chiefly finding generating functions of sequences, ODE solutions, and/or establishing nontrivial relations among them.

5.1 Generating functions

For a stream $\sigma = (r_0, r_1, ..., r_j, ...)$, we let the *ordinary generating function* [13, 22] of σ in the variable z be the power series $\mathcal{G}[\sigma](z) := \sum_{j\geq 0} r_j z^j$. We shall normally understand $\mathcal{G}[\sigma](z)$ as a formal power series, which is just another convenient, functional notation for the stream σ. When $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \mathbb{C}$, it is sometimes convenient to consider z as a real or complex variable[5]: in this case, $\mathcal{G}[\sigma](z)$ defines a (real or complex) analytic function around 0, provided that its radius of convergence is positive. In fact, we shall see that, when $\sigma = \mu_\times(p)$, then $\mathcal{G}[\sigma](z)$ is analytic. We denote by $\mathcal{G}^{-1}[g(z)]$ the inverse transformation,

[4] Python code, with instructions and examples, available at `https://local.disia.unifi.it/boreale/papers/streams.py`.

[5] For example, the study of the generating function in a complex analytic sense, in particular of its poles, provides detailed information on the asymptotic growth of the elements of σ; see [13].

mapping a power series $g(z) = \sum_{j\geq 0} r_j z^j$ back to $\sigma = (r_0, r_1, ...)$. More precisely, for any (formal or analytic) power series $g(z)$ around the origin, $\mathcal{G}^{-1}[g(z)]$ can be obtained by taking the Taylor coefficients of $g(z)$:

$$\mathcal{G}^{-1}[g(z)] = \left(\frac{g^{(0)}(0)}{0!}, \frac{g^{(1)}(0)}{1!}, \frac{g^{(2)}(0)}{2!}, ...\right) \tag{19}$$

where $g^{(j)}(z)$ denotes the j-th derivative of $g(z)$, in either formal or analytic sense. With the same convention on z, we let the *exponential* generating function of σ to be the Taylor series $\mathcal{E}[\sigma](z) := \sum_{j\geq 0} \frac{r_j}{j!} z^j$. Again, $\mathcal{E}^{-1}[g(z)]$ denotes the inverse transformation, mapping a (formal or analytic) power series $g(z)$ to the stream of its derivatives evaluated at 0:

$$\mathcal{E}^{-1}[g(z)] = (g^{(0)}(0), g^{(1)}(0), g^{(2)}(0), ...)\,. \tag{20}$$

Letting $\mathsf{fact} := (0!, 1!, 2!, ...)$ and $\exp(z) := \sum_{j\geq 0} \frac{z^j}{j!}$, the relation between $\mathcal{G}$ and $\mathcal{E}$ can be written as follows, where the Hadamard product on power series is defined as $(\sum_j a_j z^j) \odot (\sum_j b_j z^j) := \sum_j (a_j b_j) z^j$ as expected:

$$\mathcal{E}[\sigma](z) = \exp(z) \odot \mathcal{G}[\sigma](z) \tag{21}$$

$$\mathcal{E}^{-1}[g(z)] = \mathsf{fact} \odot \mathcal{G}^{-1}[g(z)]\,. \tag{22}$$

Again, for $\sigma = \mu_\otimes(p)$, we will see that $\mathcal{E}[\sigma](z)$ is analytic. The maps $\mathcal{G}[\cdot]$ and $\mathcal{E}[\cdot]$ act as $\mathbb{K}$-algebra homomorphisms between streams and functions. In particular, products of streams is transformed into product of functions[6], that is [13, 22]:

$$\mathcal{G}[\sigma \times \tau](z) = \mathcal{G}[\sigma](z) \cdot \mathcal{G}[\tau](z) \qquad\qquad \mathcal{E}[\sigma \otimes \tau](z) = \mathcal{E}[\sigma](z) \cdot \mathcal{E}[\tau](z)\,.$$

These relations allow one to transform algebraic equations on streams into algebraic equations on generating functions. One reason to perform this transformation is that, if a closed expression for the generating function can be found via analytic manipulations, the actual stream can be recovered by applying the inverse transforms (19) and (20) – that is essentially via Taylor expansion.

5.2 Algebraic streams

In what follows, we let p range over $\mathcal{P} = \mathbb{K}[x, x_1, ..., x_n]$ and $q = q(x, y)$ over $\mathbb{K}[x, y]$, while $g(z)$ still denotes a formal power series or analytic function at the origin.

▶ **Definition 5.1** (algebraic streams, [13]). *A function $g(z)$ is* algebraic *if there is a nonzero polynomial $q(x, y)$ such that $q(z, g(z))$ is identically 0. In this case, $g(z)$ is called a* branch *of $q(x, y)$. A stream σ is algebraic if $\mathcal{G}[\sigma](z)$ is algebraic.*

If the degree of $q(x, y)$ in y is k, then $q(x, y)$ has at most k branches. For example, $q(x, y) = y^2 + x - 1$ has two distinct branches, that is algebraic functions: $g(z) = \pm\sqrt{1 - z}$. When the coefficients of q are drawn from a subfield of $\mathbb{C}$, then it can be shown that the corresponding branches are also complex analytic (hence real analytic when restricted to $\mathbb{R}$); see [2]. Our starting point in the study of the connections between coalgebra morphisms and algebraic streams is the following simple result, whose easy proof relies on the fact that both μ_π and $\mathcal{G}$ are $\mathbb{K}$-algebra homomorphisms.

[6] When interpreted in a purely formal sense, hence in terms of streams: the equation for $\mathcal{G}$ just defines an alternative notation for convolution product; the equation for $\mathcal{E}$ reduces to (25).

▶ **Proposition 5.2.** *Let $p \in \mathcal{P}$ and $\sigma = \mu_\times(p)$. Suppose there is a polynomial $q(x, y) \neq 0$ such that $\mu_\times(q(x, p)) = 0$. Then $\mathcal{G}[\sigma](z)$ is a branch of q. The corresponding statement for $\mu_\otimes(\cdot)$ and $\mathcal{E}[\cdot]$ is also true.*

Pragmatically, the above result implies that, if one proves a nontrivial polynomial equation $q(x, \sigma) = 0$ for $\sigma = \mu_\pi(p)$ ($\pi \in \{\times, \otimes\}$), e.g. by using the algorithm in the previous section, then one can recover σ by Taylor expansion of one of the branches of q; see Example 5.4 below.

In the case of the convolution product ×, the result also implies that, under the given hypotheses, σ is algebraic. In fact, something more general can be said. Let the considered system of differential equations and initial conditions be $\mathcal{D} = \{\dot{x}_1 = p_1, ..., \dot{x}_n = p_n\}$ and $\rho = (r_1, ..., r_n) \in \mathbb{K}^n$, respectively; let $\sigma_i := \mu_\times(x_i)$ for $i = 1, ..., n$. As a consequence of (1), it is easy to check that the streams σ_i, hence the corresponding generating functions $\mathcal{G}[\sigma_1](z), ..., \mathcal{G}[\sigma_n](z)$, satisfy the following system of polynomial equations in the variables $x_1, ..., x_n$:

$$x_1 = r_1 + xp_1 \qquad \cdots \qquad x_n = r_n + xp_n \,. \tag{23}$$

In the terminology of Kuich and Salomaa [17, Ch.14], (23) is a *weakly strict* polynomial system (in the single letter alphabet $\{x\}$). They prove that there is a unique tuple of formal power series that solves this system, which therefore coincides with $(\sigma_1, ..., \sigma_n)$. Moreover, by invoking elimination theory, Kuich and Salomaa prove that, for each $i = 1, ..., n$, (23) implies a nontrivial polynomial equation $q(x, x_i) = 0$ for the variable x_i: see [17, Ch.16, Cor.16.11], which covers the case $\mathbb{K} = \mathbb{Q}$. We sum up the above discussion in the following.

▶ **Corollary 5.3** (algebraicity of $\mu_\times$). *Suppose that $\mathbb{K} = \mathbb{Q}$. Then, for each $p \in \mathcal{P}$, $\mu_\times(p)$ is an algebraic stream in the sense of Definition 5.1.*

When $\mathbb{K} = \mathbb{Q}$, the above result implies that $\mathcal{G}[\mu_\times(p)](z)$ is analytic. At present we do not know if the converse of this corollary is true, i.e. if all algebraic functions are expressible via polynomial SDE.

▶ **Example 5.4** (Catalan numbers). Let $\mathbb{K} = \mathbb{R}$. Consider the differential equation in one dependent variable (here $y = x_1$)

$$\dot{y} = y^2 \tag{24}$$

with the initial condition $y(0) = 1$. Let us analyse this equation from the point of view of convolution product. By (1), we have $\mu_\times(y) = \mu_\times(y)(0) + x \times (\mu_\times(y))' = 1 + x \times \mu_\times(\delta_\times(y)) = 1 + x \times \mu_\times(y^2) = 1 + x \times \mu_\times(y)^2$. Let $\sigma = \mu_\times(y)$, this leads to the polynomial equation $q(x, \sigma) = 0$, where $q(x, y) = y - xy - y^2 - 1$. Solving for y as a function of x (and renaming x to z), we obtain two branches, $y(z) = (1 \pm \sqrt{1 - 4z})/2z$. By Proposition 5.2, σ must be the series of Taylor coefficients of one or the other of these two functions. One checks that the stream obtained using the minus sign solves the equation:

$$\sigma \;=\; \mathcal{G}^{-1}\left[\frac{1 - \sqrt{1 - 4z}}{2z}\right] = (1, 1, 2, 5, 14, 42, 132, ...)\,.$$

These are the Catalan numbers, sequence A000108 in [1].

5.3 Solutions of ODEs

The shuffle product ⊗ provides a connection between streams and differential equations. A recurrent motif here is that streams and their generating functions can be used to reason on solutions of ODEs – and the other way around. In what follows, solutions might be considered in both formal and analytic sense.

When applied to $\otimes$, Proposition 5.2 may help one to recover closed forms for algebraic solutions of a ODE system, in case they exist. This is entailed by Corollary 5.6 below. In the rest of the section, we let $\mathbf{x}(z) = (x_1(z), ..., x_n(z))$ denote a solution around 0 of $\mathcal{D} = \{\dot{x}_1 = p_1, ..., \dot{x}_n = p_n\}$, considered as a system of ODEs, with the given initial conditions $\mathbf{x}(0) := \rho \in \mathbb{K}^n$. In particular, note that, when $\mathbb{K} = \mathbb{R}$, a solution always exists, is unique and analytic (Picard-Lindelöf theorem). For $p(x, x_1, ..., x_n) \in \mathcal{P}$, we let $p(z, \mathbf{x}(z))$ denote the composition of p as a function with $(z, \mathbf{x}(z))$; in turn, $p(z, \mathbf{x}(z))$ is a formal power series or analytic function around the origin. The following proposition provides a link between solutions of ODEs and shuffle product and the induced morphism, via exponential generating functions. The essential point here is that $\delta_\otimes$ coincides with Lie derivative.

▶ **Proposition 5.5.** *$p(z, \mathbf{x}(z)) = \mathcal{E}[\mu_\otimes(p)](z)$.*

When $\mathbb{K} = \mathbb{R}$, the above result implies that $\mathcal{E}[\mu_\otimes(p)](z)$ is always real analytic.

▶ **Corollary 5.6** (algebraic solutions of ODEs)**.** *Suppose that, for some nonzero $q = q(x, y)$, we have $\mu_\otimes(q(x, p)) = 0$. Then $p(z, \mathbf{x}(z))$ is a branch of $q(x, y)$.*

Proof. By Proposition 5.2, we deduce that $\mathcal{E}[\mu_\otimes(p)](z)$ is a branch of $q(x, y)$. But, by Proposition 5.5, $p(z, \mathbf{x}(z)) = \mathcal{E}[\mu_\otimes(p)](z)$. ◀

A discussion on the relation of $\mu_\otimes$ with algebraic and other classes of streams is deferred to the end of the section. We illustrate now the above results with a simple example.

▶ **Example 5.7** (factorial numbers and the solution of $\dot{y} = y^2$)**.** Consider again the equation $\dot{y} = y^2$ with $y(0) = 1$ of Example 5.4. This time we analyse this equation from the point of view of shuffle product. Let $\sigma = \mu_\otimes(y)$. Consider the polynomial $q = q(x, y) := yx - y + 1$. One checks that $q \sim 0$ in the coalgebra over $\mathcal{P}$ induced by $\delta_\otimes$: to see this, one applies the algorithm in Section 4, noting that $o(q) = q(0, 1) = 0$ and that $\delta_\otimes(q) = yq \in \langle q \rangle$. This implies $\mu_\otimes(q(x, y)) = 0$, hence, according to Proposition 5.2, $\mathcal{E}[\sigma](z)$ is a branch of $q(x, y)$. Now $q(x, y)$ defines a unique branch, $y(z) = \frac{1}{1-z}$. Then using also (22):

$$\sigma \;=\; \mathcal{E}^{-1}\left[\frac{1}{1-z}\right] = \mathsf{fact} \odot \mathcal{G}^{-1}\left[\frac{1}{1-z}\right] = \mathsf{fact} \odot (1, 1, 1, ...) = (0!, 1!, 2!, ...)\,.$$

Finally, by Corollary 5.6, the solution of (24) as an ODE with the initial condition $y(0) = 1$ is the unique branch of q, that is $y(z) = \frac{1}{1-z}$.

▶ **Example 5.8** (double factorials, again)**.** Consider again the equation $\dot{y} = y^3$ with $y(0) = 1$ of Example 4.3, and the equivalence $\mu_\otimes(q) = 0$, for $q(x, y) := y^2(x - 1/2) + 1/2$, we proved there. Let $\sigma = \mu_\otimes(y)$. According to Proposition 5.2, the exponential generating function $\mathcal{E}[\sigma](z)$ is a branch of $q(x, y)$. Now $q(x, y)$ has two branches, which are obtained by solving for y the corresponding quadratic equation. Of these, $y(z) = \sqrt{1/(1 - 2z)}$ solves the ODE and, by Proposition 5.5, is the exponential generating function of σ.

Let us also point out an interesting interplay between × and $\otimes$, that may ease compositional reasoning on streams. Depending on the equations at hand, the convolution of two streams might be more easily understood and described than their shuffle product; or a stream can be better understood in terms of the solution of an ODE. The following equality, that can be readily checked, allows one to transform convolution into stream product, and back. We let $\mathsf{fact}^{-1} := (1/0!, 1/1!, ..., 1/j!, ...)$.

$$\mathsf{fact}^{-1} \odot (\sigma \otimes \tau) = (\mathsf{fact}^{-1} \odot \sigma) \times (\mathsf{fact}^{-1} \odot \tau)\,. \tag{25}$$

We illustrate this idea with a simple example.

▶ **Example 5.9** (harmonic numbers)**.** Consider the system of two equations $\dot{y} = y^2$, $\dot{w} = y$ with initial conditions $y(0) = 1$ and $w(0) = 0$. We want to analyze this system in terms of $\otimes$. In Example 5.7, we have seen that $y(z) = \frac{1}{1-z}$ and that $\sigma := \mu_\otimes(y) = \mathsf{fact}$. We can obtain $\mu_\otimes(w)$ via Proposition 5.5 and (22), after solving the second ODE: $w(z) = \int_0^z y(u)du = \ln(\frac{1}{1-z})$, hence $\tau := \mu_\otimes(w) = \mathcal{E}^{-1}[w(z)] = \mathsf{fact} \odot (0, 1, 1/2, ..., 1/j, ...)$. To understand what $\mu_\otimes(yw) = \mu_\otimes(y) \otimes \mu_\otimes(w)$ represents, it is convenient to switch to the convolution product, by applying (25). We have

$$\begin{aligned}\mathsf{fact}^{-1} \odot \mu_\otimes(yw) &= \mathsf{fact}^{-1} \odot (\sigma \otimes \tau) = (\mathsf{fact}^{-1} \odot \sigma) \times (\mathsf{fact}^{-1} \odot \tau) \\ &= (1, 1, 1, ...) \times (0, 1, 1/2, ..., 1/j, ...) = (0, 1, 3/2, ..., \sum_{i=1}^{j} \frac{1}{i}, ...)\end{aligned}$$

which is the sequence $\alpha = (h_0, h_1, ...)$ of the harmonic numbers. Therefore $\mu_\otimes(yw) = \mathsf{fact} \odot \alpha$, and $y(z)w(z) = \frac{1}{1-z}\ln(\frac{1}{1-z}) = \mathcal{E}[\mathsf{fact} \odot \alpha](z) = \sum_{j\geq 0} h_j z^j = \mathcal{G}[\alpha](z)$ is the ordinary generating function of the harmonic numbers.

Another example of interplay between the two products arises in connection with the solutions of linear ODEs and Laplace transform; this is elaborated in the full version of the present article [11].

▶ Remark 5.10. One would like to prove for $\mu_\otimes$ a result analogous to Corollary 5.3. In this respect, let us first note that $\mu_\otimes(p)$ need not be algebraic: as we have seen in Example 5.4, $\mu_\otimes(y) = \mathsf{fact} = (0!, 1!, 2!, ...)$, which is not an algebraic stream – cf. [22], or simply note that $\mathcal{G}[\mathsf{fact}](z)$ is not analytic. The next natural candidate class to consider for inclusion is that of streams with a *holonomic* (a.k.a. *D-finite*) ordinary generating function [22]: that is, a function $y(z)$ satisfying a linear differential equation with polynomial coefficients in z. This class includes strictly algebraic streams, but $\mu_\otimes(p)$ need not be holonomic either. To see this, consider the single ODE $\dot{f} = 1 + f^2$ with $f(0) = 0$, which defines the trigonometric tangent function: $f(z) = \tan(z)$. It is known that $\sigma = \mathcal{G}^{-1}[\tan(z)]$ is not holonomic, see e.g. [18, Ch.1]. It is also known that fact is holonomic, and that the class of holonomic functions is closed under the Hadamard product [22]. Now, from Proposition 5.5 and (22), we have that: $\mu_\otimes(f) = \mathcal{E}^{-1}[\tan(z)] = \mathsf{fact} \odot \sigma$. This equality implies that $\mu_\otimes(f)$ is not holonomic, because otherwise σ would be as well. At present, we also ignore if algebraic and/or holonomic streams are included in streams obtainable via $\mu_\otimes$.

6 Conclusion

We have studied connections between polynomials, differential equations and streams, in terms of algebra and coalgebra. Our main result shows that, given any stream product that satisfies certain reasonable assumptions, there is a way to define a transition function on polynomials such that the induced unique coalgebra morphism into streams is a $\mathbb{K}$-algebra homomorphism – and vice-versa. We have applied this result to the design of a decision algorithm for polynomial stream equivalence, and to reasoning on generating functions and ordinary differential equations.

As for future work, it would be interesting to see whether we can define new notions of products that respect the format we devised in this paper. Somewhat orthogonal to this, the relation of our framework with bialgebras [16] deserves further investigation. Finally, in the field of nonlinear dynamical systems [15], convolution of discrete sequences arises as a means to describe the composition of distinct signals or subsystems (e.g., a plant and a controller); we would like to understand if our approach can be useful to reason on such systems as well.

References

1 The on-line encyclopedia of integer sequences. URL: https://oeis.org.

2 Lars V. Ahlfors. *Complex Analysis: An Introduction to the Theory of Analytic Functions of One Complex Variable (3rd edition)*. Mc Graw Hill, 1979.

3 Henning Basold, Marcello M. Bonsangue, Helle Hvid Hansen, and Jan Rutten. (co)algebraic characterizations of signal flow graphs. In *Horizons of the Mind. A Tribute to Prakash Panangaden - Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday*, volume 8464 of *LNCS*, pages 124–145. Springer, 2014. doi:10.1007/978-3-319-06880-0_6.

4 Henning Basold, Helle Hvid Hansen, Jean-Éric Pin, and Jan Rutten. Newton series, coinductively: a comparative study of composition. *Math. Struct. Comput. Sci.*, 29(1):38–66, 2019. doi:10.1017/S0960129517000159.

5 Filippo Bonchi, Marcello M. Bonsangue, Michele Boreale, Jan J. M. M. Rutten, and Alexandra Silva. A coalgebraic perspective on linear weighted automata. *Inf. Comput.*, 211:77–105, 2012. doi:10.1016/j.ic.2011.12.002.

6 Filippo Bonchi, Daniela Petrisan, Damien Pous, and Jurriaan Rot. Coinduction up-to in a fibrational setting. In *Proc. of CSL-LICS*, pages 20:1–20:9. ACM, 2014. doi:10.1145/2603088.2603149.

7 Michele Boreale. Algebra, coalgebra, and minimization in polynomial differential equations. *Log. Methods Comput. Sci.*, 15(1), 2019. doi:10.23638/LMCS-15(1:14)2019.

8 Michele Boreale. On the Coalgebra of Partial Differential Equations. In *Proc. of MFCS*, volume 138 of *LIPIcs*, pages 24:1–24:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019. doi:10.4230/LIPIcs.MFCS.2019.24.

9 Michele Boreale. Automatic pre- and postconditions for partial differential equations. In Marco Gribaudo, David N. Jansen, and Anne Remke, editors, *Proc. of QEST*, volume 12289 of *LNCS*, pages 193–210. Springer, 2020. doi:10.1007/978-3-030-59854-9_15.

10 Michele Boreale. Complete algorithms for algebraic strongest postconditions and weakest preconditions in polynomial odes. *Sci. Comput. Program.*, 193:102441, 2020. doi:10.1016/j.scico.2020.102441.

11 Michele Boreale and Daniele Gorla. Algebra and coalgebra of stream products. Full version of this work, available at *CoRR*, arXiv:2107.04455, 2021.

12 D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Undergraduate Texts in Mathematics. Springer, 2007.

13 Philippe Flajolet and Robert Sedgewick. Analytic combinatorics: functional equations, rational and algebraic functions. Research Report RR-4103, INRIA, 2001. URL: https://hal.inria.fr/inria-00072528.

14 Helle Hvid Hansen, Clemens Kupke, and Jan Rutten. Stream differential equations: Specification formats and solution methods. *Log. Methods Comput. Sci.*, 13(1), 2017. doi:10.23638/LMCS-13(1:3)2017.

15 Hassan K. Khalil. *Nonlinear Systems (3rd ed.)*. Prentice Hall, 2002.

16 Bartek Klin. Bialgebras for structural operational semantics: An introduction. *Theoretical Computer Science*, 412(38):5043–5069, 2011. doi:10.1016/j.tcs.2011.03.023.

17 W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. Monographs in Theoretical Computer Science: An EATCS Series. Springer, 1986.

18 Christian Mallinger. *Algorithmic manipulations and transformations of univariate holonomic functions and sequences*. Diplomarbeit, Johannes Kepler Universität Linz, 1996.

19 Dusko Pavlovic and M. Escardó. Calculus in coinductive form. In *Proc. of LICS*, pages 408–417. IEEE, 1998.

20 Jan J. M. M. Rutten. Behavioural differential equations: a coinductive calculus of streams, automata, and power series. *Theor. Comput. Sci.*, 308(1-3):1–53, 2003. doi:10.1016/S0304-3975(02)00895-2.

21 Jan J. M. M. Rutten. A coinductive calculus of streams. *Math. Struct. Comput. Sci.*, 15(1):93–147, 2005. doi:10.1017/S0960129504004517.

22 Richard P. Stanley. *Enumerative Combinatorics, 2nd edition.* CUP, 2012.

23 Joost Winter. *Coalgebraic Characterizations of Automata-Theoretic Classes*. PhD thesis, Radboud Universiteit Nijmegen, 2014.

Formally Verified Simulations of State-Rich Processes Using Interaction Trees in Isabelle/HOL

Simon Foster
University of York, UK

Chung-Kil Hur
Seoul National University, South Korea

Jim Woodcock
University of York, UK

Abstract

Simulation and formal verification are important complementary techniques necessary in high assurance model-based systems development. In order to support coherent results, it is necessary to provide unifying semantics and automation for both activities. In this paper we apply Interaction Trees in Isabelle/HOL to produce a verification and simulation framework for state-rich process languages. We develop the core theory and verification techniques for Interaction Trees, use them to give a semantics to the CSP and *Circus* languages, and formally link our new semantics with the failures-divergences semantic model. We also show how the Isabelle code generator can be used to generate verified executable simulations for reactive and concurrent programs.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases Coinduction, Process Algebra, Theorem Proving, Simulation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.20

Related Version *Previous Version*: `https://arxiv.org/abs/2105.05133`

Supplementary Material *Software (Source Code)*: `https://github.com/isabelle-utp/interaction-trees`; archived at `swh:1:dir:fe26a447f5611bbba15fbbc47253dba2075e1cf3`

Funding *Simon Foster*: EPSRC EP/S001190/1 (CyPhyAssure).
Jim Woodcock: EPSRC EP/V026801/1 (TAS Verifiability), EP/M025756/1 (RoboCalc).

Acknowledgements We would like to thank the anonymous reviewers of our paper, whose helpful and insightful comments have improved the content and presentation.

1 Introduction

Simulation is an important technique for prototyping system models, which is widely used in several engineering domains, notably robotics and autonomous systems [9]. For such high assurance systems, it is also necessary that controller software be formally verified, to ensure absence of faults. In order for results from simulation and formal verification to be used coherently, it is important that they are tied together using a unifying formal semantics.

Interaction trees (ITrees) have been introduced by Xia et al. [43] as a semantic technique for reactive and concurrent programming, mechanised in the Coq theorem prover. They are coinductive structures, and therefore can model infinite behaviours supported by a variety of proof techniques. Moreover, ITrees are deterministic and executable structures and so they can provide a route to both verified simulators and implementations.

Previously, we have demonstrated an Isabelle-based theory library and verification tool for reactive systems [15, 16]. This supports verification and step-wise development of nondeterministic and infinite state systems, based on the CSP [8, 21] and *Circus* [42] process languages. This includes a specification mechanism, called reactive contracts, and

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 20; pp. 20:1–20:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

calculational proof strategy. Extensions of our theory support reasoning about hybrid dynamical systems, which make it ideal for verifying autonomous robots. Recently, the set-based theory of CSP has also been mechanised [39]. However, such reactive specifications, even if deterministic, are not executable and so there is a semantic gap with implementations.

In this paper, we demonstrate how ITrees can be used as a foundation for verification and simulation of state-rich concurrent systems. For this, we present a novel mechanisation of ITrees in Isabelle/HOL, which requires substantial adaptation from the original work. The benefit is access to Isabelle's powerful proof tools, notably the *sledgehammer* automated theorem prover integration [5], but also the variety of other tools we have created in Isabelle/UTP [14], such as Hoare logic and refinement calculus [1, 30]. Isabelle's code generator allows us to automatically produce ITree-based simulations, which allows a tight development loop, where simulation and verification activities are intertwined. All our results have been mechanised, and can be found in the accompanying repository[1], and clickable icon links next to each specific result, with for Isabelle code and for Haskell code.

The structure of our paper is as follows. In §2 we show how ITrees are mechanised in Isabelle/HOL, including the core operators, and strong and weak bisimulation techniques. In §3 we show how deterministic CSP and *Circus* processes can be semantically embedded into ITrees, including operators like external choice and parallel composition. In §4 we link ITrees with the standard failures-divergences semantic model for CSP, which justifies their integration with other CSP-based techniques. In §5 we show how the code generator can be used to generate simulations. In §6 we briefly consider related work, and in §7 we conclude.

2 Interaction Trees in Isabelle/HOL

Here, we introduce Interaction Trees (ITrees) and develop the main theory in Isabelle/HOL, along with several novel results. ITrees were originally mechanised in Coq by Xia et al. [43]. Our mechanisation in Isabelle/HOL brings unique advantages, including a flexible frontend syntax, an array of automated proof tools, and code generation to several languages.

ITrees are potentially infinite trees whose edges are decorated with events, representing the interactions between a process and its environment. They are parametrised over two sorts (types): E of events and R of return values (or states). There are three possible interactions: (1) termination, returning a value in R; (2) an internal event (τ); or (3) a choice between several visible events. In Isabelle/HOL, we encode ITrees using a codatatype [4, 7]:

▶ **Definition 1** (Interaction Tree Codatatype).

```
codatatype ('e, 'r) itree =
  Ret 'r | Sil "('e, 'r) itree" |  Vis "'e ⇸ ('e, 'r) itree"
```

Type parameters `'e` and `'r` encode the sorts E and R. Constructor *Ret* represents a return value, and *Sil* an internal event, which evolves to a further ITree. A visible event choice (*Vis*) is represented by a partial function ($A \pfun B$) from events to ITrees, with a potentially infinite domain. This representation is the main deviation from ITrees in Coq [43] (see §6). Here, $A \pfun B$ is isomorphic to `A ⇒ B option`, where `B option` can take the value `None` or `Some x` for `x::B`. We usually specify partial functions using $\lambda\, x \in A \bullet f(x)$, which restricts a function f to the domain A. We write $\{\mapsto\}$ for an empty function, and adopt several operators from the Z notation [38], such as dom, override ($F \oplus G$), and domain restriction ($A \lhd F$). With the associated theorems, we can use Isabelle's simplifier to equationally calculate the domain and other properties of choice partial functions, which provides a high degree of proof automation.

[1] `https://github.com/isabelle-utp/interaction-trees`

We sometimes use $\checkmark_v$ to denote $\mathit{Ret}\, v$, τP to denote $\mathit{Sil}\, P$, and $\Box\, e \in E \to P(e)$ to denote $\mathit{Vis}(\lambda\, e \in E \bullet P(e))$, which are more concise and suggestive of their process algebra equivalents. We write $e_1 \to P_1 \Box \cdots \Box\, e_n \to P_n$ when $E = \{e_1, \cdots, e_n\}$. We use $\tau^n P$ for an ITree prefixed by $n \in \mathbb{N}$ internal events. We define $\mathit{stop} \triangleq \mathit{Vis}\,\{\mapsto\}$, a deadlock situation where no event is possible. An example is $a \to \tau(\checkmark_x) \Box\, b \to \mathit{stop}$, which can either perform an a followed by a τ, and then terminate returning x, or perform a b and then deadlock.

We call an ITree *unstable* if it has the form τP, and *stable* otherwise. An ITree stabilises, written $P \Downarrow$, if it becomes stable after a finite sequence of τ events, that is $\exists\, n\, P' \bullet P = \tau^n P' \wedge \mathit{stable}(P')$. An ITree that does not stabilise is divergent, written $P \Uparrow \triangleq \neg(P \Downarrow)$.

Using the operators mentioned so far, we can specify only ITrees of finite depth. Infinite ITrees can be specified using primitive corecursion [4], as exemplified below.

```
primcorec div :: "(’e, ’s) itree" where "div = τ div"
primcorec run :: "’e set ⇒ (’e, ’s) itree" where
  "run E = Vis (map_pfun (λ x. run E) (pId_on E))"
```

The `primcorec` command requires that every corecursive call on the right-hand side of an equation is guarded by a constructor. ITree *div* represents the divergent ITree that does not terminate, and only performs internal activity. It is divergent, $\mathit{div} \Uparrow$, since it never stabilises. Moreover, we can show that *div* is the unique fixed-point of τ^{n+1} for any $n \in \mathbb{N}$, $\tau^{n+1} P = P \Leftrightarrow P = \mathit{div}$, and consequently *div* is the only divergent ITree: $P \Uparrow\; \Rightarrow P = \mathit{div}$.

ITree $\mathit{run}\, E$ can repeatedly perform any $e \in E$ without ceasing. It has the equivalent definition of $\mathit{run}\, E \triangleq \Box e \in E \to \mathit{run}\, E$, and thus the special case $\mathit{run}\, \emptyset = \mathit{stop}$. The formulation above uses the function `map_pfun` :: (`’b`$\Rightarrow$`’c`) $\Rightarrow$ (`’a`$\nrightarrow$`’b`) $\Rightarrow$ (`’a`$\nrightarrow$`’c`) which maps a total function over every output of a partial function. Function `pId_on E` is the identity partial function with domain `E`. This formulation is required to satisfy the syntactic guardedness requirements. For the sake of readability, we elide these details in the definitions that follow.

Corecursive definitions can have several equations ordered by priority, like a recursive function. We specify a monadic bind operator for ITrees [43] using such a set of equations.

▶ **Definition 2** (Interaction Tree Bind). *We fix $P, P' : (E, R)\mathit{itree}$, $K : R \Rightarrow (E, S)\mathit{itree}$, $r : R$, and $F : E \nrightarrow (E, S)\mathit{itree}$. Then, $P \gg\!= K$ is defined corecursively by the equations*

$$\checkmark_r \gg\!= K = K\, r \qquad \tau P' \gg\!= K = \tau(P' \gg\!= K) \qquad \mathit{Vis}\, F \gg\!= K = \mathit{Vis}\,(\lambda\, e \in \mathrm{dom}(F) \bullet F(x) \gg\!= K)$$

The intuition of $P \gg\!= K$ is to execute P, and whenever it terminates ($\checkmark_x$), pass the given value x on to the continuation K. We term K a Kleisli tree [43], or KTree, since it is a Klesli lifting of an ITree. KTrees are of great importance for defining processes that depend on a previous state. For this, we define the type synonym $(E, S)\mathit{htree} \triangleq (S \Rightarrow (E, S)\mathit{itree})$ for a homogeneous KTree. We define the Kleisli composition operator $P \fatsemi Q \triangleq (\lambda\, x.Px \gg\!= Q)$, so symbolised because it is used as sequential composition. Bind satisfies several algebraic laws:

▶ **Theorem 3** (Interaction Tree Bind Laws).

$$\begin{array}{cc}
\mathit{Ret}\, x \gg\!= K = K\, x & \mathit{Ret} \fatsemi K = K \\
P \gg\!= \mathit{Ret} = P & K \fatsemi \mathit{Ret} = K \\
P \gg\!= (\lambda\, x.(Q\, x \gg\!= R)) = (P \gg\!= Q) \gg\!= R & K_1 \fatsemi (K_2 \fatsemi K_3) = (K_1 \fatsemi K_2) \fatsemi K_3 \\
\mathit{div} \gg\!= K = \mathit{div} & \mathit{run}\, E \gg\!= K = \mathit{run}\, E
\end{array}$$

Bind satisfies the three monad laws: it has *Ret* as left and right units, and is essentially associative. Moreover, both *div* and *run* are left annihilators for bind, since they do not terminate. From the monad laws, we can show that $(\fatsemi, \mathit{Ret})$ also forms a monoid.

The laws of Theorem 3 are proved by coinduction, using the following derivation rule.

▶ **Theorem 4** (ITree Coinduction). *We fix a relation $\mathcal{R} : (E, R)\mathit{itree} \leftrightarrow (E, R)\mathit{itree}$ and then given $(P, Q) \in \mathcal{R}$ we can deduce $P = Q$ provided that the following conditions of $\mathcal{R}$ hold:*

$$\forall (P', Q') \in \mathcal{R} \bullet \mathit{is\_Ret}(P') = \mathit{is\_Ret}(Q') \wedge \mathit{is\_Sil}(P') = \mathit{is\_Sil}(Q') \wedge \mathit{is\_Vis}(P') = \mathit{is\_Vis}(Q');$$
$$\forall (x, y) \bullet (\mathit{Ret}\, x, \mathit{Ret}\, y) \in \mathcal{R} \Rightarrow x = y;$$
$$\forall (P', Q') \bullet (\mathit{Sil}\, P', \mathit{Sil}\, Q') \in \mathcal{R} \Rightarrow (P', Q') \in \mathcal{R};$$
$$\forall (F, G) \bullet (\mathit{Vis}\, F, \mathit{Vis}\, G) \in \mathcal{R} \Rightarrow (\mathrm{dom}(F) = \mathrm{dom}(G) \wedge (\forall\, e \in \mathrm{dom}(F) \bullet (F(e), G(e)) \in \mathcal{R}))$$

To show $P = Q$, we need to construct a (strong) bisimulation $\mathcal{R}$ and show that $(P, Q) \in \mathcal{R}$. There are four provisos to show that $\mathcal{R}$ is a bisimulation. The first requires that only ITrees of the same kind are related, where *is_Ret*, *is_Sil*, and *is_Vis* distinguish the three cases. The second proviso states that if $(\checkmark_x, \checkmark_y) \in \mathcal{R}$ then $x = y$. The third proviso states that internal events must yield bisimilar continuations: $(\tau P, \tau Q) \in \mathcal{R} \Rightarrow (P, Q) \in \mathcal{R}$. The final proviso states that for two visible interactions the two functions must have the same domain $(\mathrm{dom}(F) = \mathrm{dom}(G))$ and every event $e \in \mathrm{dom}(F)$ must lead to bisimilar continuations. The majority of our ITree proofs in Isabelle apply this law, and then use a mixture of equational simplification and automated reasoning with *sledgehammer* to discharge the resulting provisos.

Next, we define an operator for iterating ITrees:

```
corec while :: "('s ⇒ bool) ⇒ ('e, 's) htree ⇒ ('e, 's) htree" where
"while b P s = (if (b s) then Sil (P s ⋙= while b P) else Ret s)"
```

This is not primitively corecursive, since the corecursive call uses $\ggg=$, and so we define it using the **corec** command [6, 3] instead of **primcorec**. This requires us to show that $\ggg=$ is a "friendly" corecursive function [3]: it consumes at most one input constructor to produce one output constructor. A while loop iterates whilst the condition b is satisfied by state s. In this case, a τ event is followed by the loop body and the corecursive call. If the condition is false, the current state is returned. We introduce the special cases $\mathit{loop}\, F \triangleq \mathit{while}\, (\lambda\, s \bullet \mathit{True})\, F$ and $\mathit{iter}\, P \triangleq \mathit{loop}\, (\lambda\, s \bullet P)\, ()$, which represent infinite loops with and without state, respectively. We can show that $\mathit{iter}\, (\checkmark_{()}) = \mathit{div}$, since it never terminates and has no visible behaviour.

Though strong bisimulation is a useful equivalence, we often wish to abstract over τs. We therefore also introduce weak bisimulation, $P \approx Q$, as a coinductive-inductive predicate. It requires us to construct a relation $\mathcal{R}$ such that whenever (P, Q) in $\mathcal{R}$ both stabilise, all their visible event continuations are also related by $\mathcal{R}$. For example, $\tau^m\, P \approx \tau^n\, Q$ whenever $P \approx Q$. We have proved that $\approx$ is an equivalence relation, and $P \approx \mathit{div} \Rightarrow P = \mathit{div}$.

3 CSP and Circus

Here, we give an ITree semantics to deterministic fragments of the CSP [8, 21] and *Circus* [42, 32] languages. Our deterministic CSP fragment is consistent with the one identified by Roscoe [36, Section 10.5]. The standard CSP denotational semantics is provided by the failures-divergences model [8, 36], and we provide preliminary results on linking to this in §4.

3.1 CSP

CSP processes are parametrised by an event alphabet (Σ), which specifies the possible ways a process communicates with its environment. For ITrees, Σ is provided by the type parameter E. Whilst E is typically infinite, it is usually expressed in terms of a finite set of channels, which can carry data of various types. Here, we characterise channels abstractly using prisms [33], a concept well known in the functional programming world:

▶ **Definition 5** (Prisms). *A prism is a quadruple* $(\mathcal{V}, \Sigma, \mathit{match}, \mathit{build})$ *where* $\mathcal{V}$ *and* Σ *are non-empty sets. Functions* $\mathit{match} : \Sigma ⇸ \mathcal{V}$ *and* $\mathit{build} : \mathcal{V} \Rightarrow \Sigma$ *satisfy the following laws:*

$$\mathit{match}(\mathit{build}\, x) = x \qquad\qquad y \in \mathrm{dom}(\mathit{match}) \Rightarrow \mathit{build}\,(\mathit{match}\, y) = y$$

We write $X : V \xrightarrow{\triangle} E$ *if* X *is a prism with* $\Sigma_X = E$ *and* $\mathcal{V}_X = V$.

Intuitively, a prism abstractly characterises a datatype constructor, E, taking a value of type $\mathcal{V}$. Then, *build* is the constructor, and *match* is the destructor, which is partial due to the possibility of several disjoint constructors. For CSP, each prism models a channel in E carrying a value of type $\mathcal{V}$. We have created a command **chantype**, which automates the creation of prism-based event alphabets.

CSP processes typically do not return data, though their components may, and so they are typically denoted as ITrees of type $(E, ())\,\mathit{itree}$, returning the unit type $()$. An example is $\mathit{skip} \triangleq \mathit{Ret}\,()$, which is a degenerate form of *Ret*. We now define the basic CSP operators.

▶ **Definition 6** (Basic CSP Constructs).

$$\begin{aligned}
\mathit{inp} &:: (V \xrightarrow{\triangle} E) \Rightarrow V\ \mathit{set} \Rightarrow (E, V)\,\mathit{itree} \\
\mathit{inp}\, c\, A &\triangleq \mathit{Vis}\,(\lambda\, e \in \mathrm{dom}(\mathit{match}_c) \cap \mathit{build}_c⦇\, A\, ⦈ \bullet \mathit{Ret}\,(\mathit{match}_c\, e))
\end{aligned}$$

$$\begin{aligned}
\mathit{outp} &:: (V \xrightarrow{\triangle} E) \Rightarrow V \Rightarrow (E, ())\,\mathit{itree} & \mathit{guard}\, b &:: \mathbb{B} \Rightarrow (E, ())\,\mathit{itree} \\
\mathit{outp}\, c\, v &\triangleq \mathit{Vis}\,\{\mathit{build}_c\, v \mapsto \mathit{Ret}\,()\} & \mathit{guard}\, b &\triangleq (\mathit{if}\, b\ \mathit{then}\ \mathit{skip}\ \mathit{else}\ \mathit{stop})
\end{aligned}$$

An input event $(\mathit{inp}\, c\, A)$ permits any event over the channel c, that is $e \in \mathrm{dom}(\mathit{match}_c)$, provided that its parameter is in A ($e \in \mathit{build}_c⦇\, A\, ⦈$), and it returns the value received for use by a continuation. It corresponds to the `trigger` construct in [43]. An output event $(\mathit{outp}\, c\, v)$ permits a single event, v on channel c, and returns a null value of type $()$. We also define the special case $\mathit{sync}\, e \triangleq \mathit{outp}\, e\,()$ for a basic event $e :: () \xrightarrow{\triangle} E$. A $\mathit{guard}\, b$ behaves as *skip* if $b = \mathit{true}$ and otherwise deadlocks. It corresponds to the guard in CSP, which can be defined as $b \,\&\, P \triangleq (\mathit{guard}\, b \ggg\!= (\lambda\, x \bullet P))$.

Using the monadic "do" notation, which boils down to applications of $\ggg\!=$, we can now write simple reactive programs such as $\mathit{do}\{x \leftarrow \mathit{inp}\, c;\ \mathit{outp}\, d\,(2 \cdot x);\ \mathit{Ret}\, x\}$, which inputs x over channel $c : \mathbb{N} \xrightarrow{\triangle} E$, outputs $2 \cdot x$ over channel d, and finally terminates, returning x.

Next, we define the external choice operator, $P \,\Box\, Q$, where the environment resolves the choice with an initial event of P or Q. In CSP, $\Box$ can also introduce nondeterminism, for example $(a \rightarrow P) \,\Box\, (a \rightarrow Q)$ introduces an internal choice, since the a event can lead to P or Q, and is equal to $a \rightarrow (P \sqcap Q)$. Since we explicitly wish to avoid introducing such nondeterminism, we make a design choice to exclude this possibility by construction. There are other possibilities for handling nondeterminism in ITrees, which we consider in §7. As for $\ggg\!=$, we define external choice corecursively using a set of ordered equations.

▶ **Definition 7** (External choice). $P \,\Box\, Q$, *is defined by the following set of equations:*

$$\begin{aligned}
(\mathit{Vis}\, F) \,\Box\, (\mathit{Vis}\, G) &= \mathit{Vis}\,(F \odot G) & (\mathit{Ret}\, x) \,\Box\, (\mathit{Vis}\, G) &= \mathit{Ret}\, x \\
(\mathit{Sil}\, P') \,\Box\, Q &= \mathit{Sil}\,(P' \,\Box\, Q) & (\mathit{Vis}\, F) \,\Box\, (\mathit{Ret}\, y) &= \mathit{Ret}\, y \\
P \,\Box\, (\mathit{Sil}\, Q') &= \mathit{Sil}\,(P \,\Box\, Q') & (\mathit{Ret}\, x) \,\Box\, (\mathit{Ret}\, y) &= (\mathit{if}\, x = y\ \mathit{then}\,(\mathit{Ret}\, x)\ \mathit{else}\ \mathit{stop})
\end{aligned}$$

where $F \odot G \triangleq (\mathrm{dom}(G) \mathbin{\triangleleft\!\!\!-} F) \oplus (\mathrm{dom}(F) \mathbin{\triangleleft\!\!\!-} G)$

An external choice between two functions F and G essentially combines all the choices presented using $F \odot G$. The caveat is that if the domains of F and G overlap, then any events in common are excluded. Thus, $\odot$ restricts the domain of F to maplets $e \mapsto P$

where $e \notin \mathrm{dom}(G)$, and vice-versa. This has the effect that $(a \to P) \Box (a \to Q) = \mathit{stop}$, for example. In the special case that $\mathrm{dom}(F) \cap \mathrm{dom}(G) = \emptyset$, $P \odot Q = P \oplus Q$. We chose this behaviour to ensure that $\Box$ is commutative, though we could alternatively bias one side.

Internal steps on either side of $\Box$ are greedily consumed. Due to the equation order, τ events have the highest priority, following a maximal progress assumption [20]. Return events also have priority over visible events. If two returns are present then they must agree on the value, otherwise they deadlock. External choice satisfies several important properties:

▶ **Theorem 8** (External Choice Properties).

$$P \Box Q = Q \Box P \quad \mathit{stop} \Box P = P \quad \mathit{div} \Box P = \mathit{div} \quad P \Box (\tau^n\, Q) = (\tau^n\, P) \Box Q = \tau^n (P \Box Q)$$

$$(\mathit{Vis}\, F \Box \mathit{Vis}\, G) \ggg H = (\mathit{Vis}\, F \ggg H) \Box (\mathit{Vis}\, G \ggg H)$$

External choice is commutative and has *stop* as a unit. It has *div* as an annihilator, because the τ events means that no other activity is chosen. A finite number of τ events on either the left or right can be extracted to the front. Finally, bind distributes from the left across a visible event choice. We prove these properties using coinduction (Theorem 4), followed by several invocations of *sledgehammer* to discharge the resulting provisos.

Using the operators defined so far, we can implement a simple buffer process:

```
chantype Chan = Input::integer  Output::integer  State::"integer list"

definition buffer :: "integer list ⇒ (Chan, integer list) itree" where
"buffer = loop (λ s.
                   do { i ← inp Input {0..}; Ret (s @ [i]) }
                 □ do { guard(length s > 0); outp Output (hd s); Ret (tl s) }
                 □ do { outp State s; Ret s })"
```

We first create a channel type `Chan`, which has channels (prisms) for inputs and outputs, and to view the current buffer state. We define the buffer process as a simple loop with a choice with three branches inside. The variable `s::integer list` denotes the state. The first branch allows a value to be received over `Input`, and then returns `s` with the new value added, and then iterates. The second branch is only active when the buffer is not empty. It outputs the head on `Output`, and then returns the tail. The final branch simply outputs the current state. In §5 we will see how such an example can be simulated.

Next, we tackle parallel composition. The objective is to define the usual CSP operator $P \llbracket E \rrbracket Q$, which requires that P and Q synchronise on the events in E and can otherwise evolve independently. We first define an auxiliary operator for merging choice functions.

$$\begin{aligned} \mathit{merge}_E(F, G) = {} & (\lambda\, e \in \mathrm{dom}(F) \setminus (\mathrm{dom}(G) \cup E) \bullet \mathit{Left}(F(e))) \\ & \oplus (\lambda\, e \in \mathrm{dom}(G) \setminus (\mathrm{dom}(F) \cup E) \bullet \mathit{Right}(G(e))) \\ & \oplus (\lambda\, e \in \mathrm{dom}(F) \cap \mathrm{dom}(G) \cap E \bullet \mathit{Both}(F(e), G(e)) \end{aligned}$$

Operator $\mathit{merge}_E(F, G)$ merges two event functions. Each event is tagged depending on whether it occurs on the *Left*, *Right*, or *Both* sides of a parallel composition. An event in $\mathrm{dom}(F)$ can occur independently when it is not in E, and also not in $\mathrm{dom}(G)$. The latter proviso is required, like for $\Box$, to prevent nondeterminism by disallowing the same event from occurring independently on both sides. An event in $\mathrm{dom}(G)$ can occur independently through the symmetric case for $\mathrm{dom}(F)$. An event can synchronise provided it is in the domain of both choice functions and the set E. We use this operator to define generalised parallel composition. For the sake of presentation, we present partial functions as sets.

▶ **Definition 9.** *$P \parallel_E Q$ is defined corecursively by the following equations:*

$$(\mathit{Vis}\, F) \parallel_E (\mathit{Vis}\, G) = \mathit{Vis} \begin{pmatrix} \{e \mapsto (P' \parallel_E (\mathit{Vis}\, G)) \mid (e \mapsto \mathit{Left}(P')) \in \mathit{merge}_A(F, G)\} \\ \oplus \{e \mapsto ((\mathit{Vis}\, F) \parallel_E Q') \mid (e \mapsto \mathit{Right}(Q')) \in \mathit{merge}_E(F, G)\} \\ \oplus \{e \mapsto (P' \parallel_E Q') \mid (e \mapsto \mathit{Both}(P', Q')) \in \mathit{merge}_E(F, G)\} \end{pmatrix}$$

$$(\mathit{Sil}\, P') \parallel_E Q = \mathit{Sil}\,(P' \parallel_E Q) \qquad P \parallel_E (\mathit{Sil}\, Q') = \mathit{Sil}\,(P \parallel_E Q')$$

$$(\mathit{Ret}\, x) \parallel_E (\mathit{Ret}\, y) = \mathit{Ret}\,(x, y)$$

$$(\mathit{Ret}\, x) \parallel_E (\mathit{Vis}\, G) = \mathit{Vis}\,\{e \mapsto \mathit{Ret}\, x \parallel_E Q' \mid (e \mapsto Q') \in G\}$$

$$(\mathit{Vis}\, F) \parallel_E (\mathit{Ret}\, y) = \mathit{Vis}\,\{e \mapsto P' \parallel_E \mathit{Ret}\, y \mid (e \mapsto P') \in F\}$$

The most complex case is for *Vis*, which constructs a new choice function by merging F and G. The three cases are again represented by three partial functions. The first two allow the left and right to evolve independently to P' and Q', respectively, using one of their enabled events, leaving their opposing side, *Vis* G and *Vis* F respectively, unchanged. The third case allows them both to evolve simultaneously on a synchronised event.

The *Sil* cases allow τ events to happen independently and with priority. If both sides can return a value, x and y, respectively then the parallel composition returns a pair, which can later be merged if desired. The final two cases show what happens when only one side has a return value, and the other side has visible events. In this case, the *Ret* value is retained and pushed through the parallel composition, until the other side also terminates.

We use $\parallel_E$ to define two special cases for CSP: $P \,[\![E]\!]\, Q \triangleq (P \parallel_E Q) \ggg\!= (\lambda(x, y) \bullet \mathit{Ret}\,())$ and $P \mathbin{|||} Q \triangleq P \,[\![\emptyset]\!]\, Q$. As usual in CSP, these operators do not return any values and so $P, Q :: (E, ())\mathit{itree}$. The $P \,[\![E]\!]\, Q$ operator is similar to $\parallel_E$, except that if both sides terminate any resultant values are discarded and a null value is returned. This is achieved by binding to a simple merge function. P and Q do not return values, and so this has no effect on the behaviour, just the typing. The interleaving operator $P \mathbin{|||} Q$, where there is no synchronisation, is simply defined as $P \,[\![\emptyset]\!]\, Q$. We prove several algebraic laws:

$$(P \parallel_E Q) = (Q \parallel_E P) \ggg\!= (\lambda(x, y) \bullet \mathit{Ret}\,(y, x)) \qquad \mathit{div} \parallel_E P = \mathit{div}$$

$$P \,[\![E]\!]\, Q = Q \,[\![E]\!]\, P \qquad P \mathbin{|||} Q = Q \mathbin{|||} P \qquad \mathit{skip} \mathbin{|||} P = P$$

Parallel composition is commutative, except that we must swap the outputs, and so $[\![E]\!]$ and $|||$ are commutative as well. Parallel has *div* as an annihilator for similar reasons to $\Box$. For $|||$, *skip* is a unit since there is no possibility of communication and no values are returned.

The final operator we consider is hiding, $P \setminus A$, which turns the events in A into τs:

▶ **Definition 10** (Hiding)**.** *$P \setminus A$ is defined corecursively by the following equations:*

$$\mathit{Vis}(F) \setminus A = \begin{cases} \mathit{Sil}\,(F(e) \setminus A) & \text{if } A \cap \mathrm{dom}(F) = \{e\} \\ \mathit{Vis}\,\{(e, P \setminus A) \mid (e, P) \in F\} & \text{if } A \cap \mathrm{dom}(F) = \emptyset \\ \mathit{stop} & \text{otherwise} \end{cases}$$

$$\mathit{Sil}(P) \setminus A = \mathit{Sil}(P \setminus A) \qquad \mathit{Ret}\, x \setminus A = \mathit{Ret}\, x$$

We consider a restricted version of hiding where only one event can be hidden at a time, to avoid nondeterminism. When hiding the events of A in the choice function F there are three cases: (1) there is precisely one event $e \in A$ enabled, in which case it is hidden; (2) no enabled event is in A, in which case the event remains visible; (3) more than one $e \in A$ is enabled, and so we deadlock. We again impose maximal progress here, so that an enabled event to be hidden is prioritised over other visible events: $(a \rightarrow P \mathrel{[\!]} b \rightarrow Q) \setminus \{a\} = \tau P$, for example. In spite of the significant restrictions on hiding, it supports the common pattern where one output event is matched with an input event. Moreover, a priority can be placed on the order in which events are hidden, rather than deadlocking, by sequentially hiding events. Hiding can introduce divergence, as the following theorem shows: $(\mathsf{iter}\,(\mathsf{sync}\;e)) \setminus e = \mathsf{div}$.

3.2 Circus

Whilst CSP processes can be parametrised to allow modelling state, there is no support for explicit state operators like assignment. The *do* notation somewhat allows variables, but these are immutable and are not preserved across iterations. *Circus* [42, 32] is an extension of CSP that allows state variables. Given a state variable `buf::integer list`, the buffer example can be expressed in *Circus* as follows:

$$
\begin{aligned}
&\mathit{buf} := [\,] \mathbin{;} \mathit{loop}((\mathit{Input}?(i) \rightarrow \mathit{buf} := \mathit{buf} \mathbin{@} [i])\\
&\qquad\qquad\qquad \Box\; ((\mathit{length}(\mathit{buf}) > 0) \;\&\; \mathit{Output}!(\mathit{hd}\,\mathit{buf}) \rightarrow \mathit{buf} := \mathit{tl}\,\mathit{buf})\\
&\qquad\qquad\qquad \Box\; \mathit{State}!(\mathit{buf}) \rightarrow \mathsf{Skip})
\end{aligned}
$$

We update the state with assignments, which are threaded through sequential composition.

In our work [15, 14, 16], each state variable is modelled as a lens [12], $x :: \mathcal{V} \Longrightarrow \mathcal{S}$. This is a pair of functions $\mathsf{get} :: \mathcal{V} \Rightarrow \mathcal{S}$ and $\mathsf{put} :: \mathcal{S} \Rightarrow \mathcal{V} \Rightarrow \mathcal{S}$, which query and update the variables present in state $\mathcal{S}$, and satisfy intuitive algebraic laws [14]. They allow an abstract representation of state spaces, where no explicit model is required to support the laws of programming [22]. Lenses can be designated as independent, $x \bowtie y$, meaning they refer to different regions of $\mathcal{S}$. An expression on state variables is simply a function $e :: \mathcal{S} \Rightarrow \mathcal{V}$, where $\mathcal{V}$ is the return type. We can check whether an expression e uses a lens x using unrestriction, written $x \sharp e$. If $x \sharp e$, then e does not use x in its valuation, for example $x \sharp (y + 1)$, when $x \bowtie y$. Updates to variables can be expressed using the notation $[x_1 \rightsquigarrow e_1, x_2 \rightsquigarrow e_2, \cdots]$, with $x_i :: \mathcal{V}_i \Longrightarrow \mathcal{S}$ and $e_i :: \mathcal{S} \Rightarrow \mathcal{V}_i$, which represents a function $\mathcal{S} \Rightarrow \mathcal{S}$.

We can characterise *Circus* through a Kleisli lifting of CSP processes that return values, so that *Circus* actions are simply homogeneous KTrees. We define the core operators below:

▶ **Definition 11** (Circus Operators).

$$
\begin{aligned}
\langle \sigma \rangle &\triangleq (\lambda\, s \bullet \mathsf{Ret}(\sigma(s)))\\
x := e &\triangleq \langle [x \rightsquigarrow e] \rangle\\
c?x{:}A \rightarrow F(x) &\triangleq (\lambda\, s \bullet \mathsf{inp}\; c\; A \ggg\!= (\lambda\, x \bullet F(x)\; s))\\
c!e \rightarrow P &\triangleq (\lambda\, s \bullet \mathsf{outp}\; c\; (e\; s) \ggg\!= (\lambda\, x \bullet P\; s))\\
P \Box Q &\triangleq (\lambda\, s \bullet P(s) \Box Q(s))\\
P \llbracket ns_1 | E | ns_2 \rrbracket\, Q &\triangleq (\lambda\, s \bullet (P(s) \parallel_E Q(s)) \ggg\!= (\lambda (s_1, s_2) \bullet s \lhd_{ns_1} s_1 \lhd_{ns_2} s_2))
\end{aligned}
$$

Operator $\langle \sigma \rangle$ lifts a function $\sigma : \mathcal{S} \Rightarrow \mathcal{S}$ to a KTree. It is principally used to represent assignments, which can be constructed using our maplet notation, such that a single assignment $x := e$ is $\langle [x \rightsquigarrow e] \rangle$. Most of the remaining operators are defined by lifting of their CSP equivalents. An output $c!e \rightarrow P$ carries an expression e, rather than a value, which can depend on the state variables. The main complexity is the *Circus* parallel operator, $P \llbracket ns_1 | E | ns_2 \rrbracket\, Q$, which allows P and Q to act on disjoint portions of the state, characterised by the name sets ns_1 and ns_2. We represent ns_1 and ns_2 as independent lenses, $ns_1 \bowtie ns_2$, though they can be thought of as sets of variables with $ns_1 \cap ns_2 = \emptyset$. The definition of the operator first lifts $\|_E$, and composes this with a merge function. The merge function constructs a state that is composed of the ns_1 region from the final state of P, the ns_2 region from Q, and the remainder coming from the initial state s. This is achieved using the lens override operator $s_1 \triangleleft_X s_2$, which extracts the region described by X from s_2 and overwrites the corresponding region in s_1, leaving the complement unchanged.

Our *Circus* operators satisfy many standard laws [32, 16], beyond the CSP laws:

$$\begin{aligned}
\langle\sigma\rangle \fatsemi \langle\rho\rangle &= \langle\rho \circ \sigma\rangle \\
\langle\sigma\rangle \fatsemi (P \mathbin{\Box} Q) &= (\langle\sigma\rangle \fatsemi P) \mathbin{\Box} (\langle\sigma\rangle \fatsemi Q) \\
x := e \fatsemi y := f &= y := f \fatsemi x := e && \text{if } x \bowtie y, x \sharp f, y \sharp e \\
P \llbracket ns_1 | E | ns_2 \rrbracket Q &= Q \llbracket ns_2 | E | ns_1 \rrbracket P && \text{if } ns_1 \bowtie ns_2
\end{aligned}$$

Sequential composition of two state updates σ and ρ entails their functional composition. State updates distribute through external choice from the left. Two variable assignments commute provided their variables are independent ($x \bowtie y$) and their respective expressions do not depend on the adjacent variable. *Circus* parallel composition is commutative, provided that we also switch the name sets.

4 Linking to Failures-Divergences Semantics

Next, we show how ITrees are related to the standard failures-divergences semantics of CSP [8]. The utility of this link is to both allow symbolic verification of ITrees and allow them to act as a target of step-wise refinement. In this way, we can use existing the mechanisations of the CSP set-based and relational semantics [39, 16] to capture and reason about nondeterministic specifications, and use ITrees to provide executable implementations.

In the failures-divergences model, a process is characterised by two sets: $F :: (E^{\checkmark}\,\mathit{list} \times E\,\mathit{set})\,\mathit{set}$ and $D :: \mathbb{P}(E\,\mathit{list})$, which are, respectively, the set of failures and divergences. A failure is a trace of events plus a set of events that can be refused at the end of the interaction. A divergence is a trace of events that leads to divergent behaviour. A distinguished event $\checkmark \in E$ is used as the final element of a trace to indicate that this is a terminating observation.

For example, consider the process $a \to c \to \mathit{skip} \mathbin{\Box} b \to \mathit{div}$, which initially permits an a or b event, and following a permits a c event. It exhibits the failure $([], \{c\})$, since before any events are performed, the event c is being refused. A second failure is $([a], \{a, b\})$, since after performing an a, only c is enabled and the other events are refused. A third failure is $([a, c, \checkmark], \{a, b, c\})$, which represents successful termination, after which all events are refused. This process also has a divergence trace $[b]$, since after performing event b, the process diverges. If a divergent state is unreachable then D is empty. Here, we show how to extract F and D from any ITree, and thus processes constructed from the operators of §3.

We begin by giving a big-step operational semantics to ITrees, using an inductive predicate.

▶ **Definition 12** (Big-Step Operational Semantics).

$$\frac{-}{P \xrightarrow{[]} P} \qquad \frac{P \xrightarrow{tr} P'}{\tau P \xrightarrow{tr} P'} \qquad \frac{e \in E \quad F(e) \xrightarrow{tr} P'}{(\llbracket\, x \in E \bullet F(x)) \xrightarrow{e \# tr} P'}$$

The relation $P \xrightarrow{tr} Q$ means that P can perform the trace of visible events contained in the list $tr : E\,\mathit{list}$ and evolve to the ITree Q. This relation skips over τ events. The first rule states that any ITree may perform an empty trace ($[]$) and remain at the same state. The second rule states that if P can evolve to P' by performing tr, then so can τP. The final rule states that if e is an enabled visible event, and $P(e)$ can evolve to P' by doing tr, then the event choice can evolve to P' via $e\#tr$, which is tr with e inserted at the head. This inductive predicate is different from the trace predicate (`is_trace_of`) in [43], since $P \xrightarrow{tr} P'$ records both the trace and the continuation ITree. It is therefore more general, and provides the foundation for characterising both structural operational and denotational semantics. With these laws, we can prove the usual operational laws for sequential composition as theorems:

▶ **Theorem 13** (Sequential Operational Semantics).

$$\frac{-}{\mathit{skip} \longrightarrow \checkmark_{()}} \qquad \frac{P \xrightarrow{tr} P'}{(P \gg\!= Q) \xrightarrow{tr} (P' \gg\!= Q)} \qquad \frac{P \xrightarrow{tr_1} \checkmark_x \quad Q(x) \xrightarrow{tr_2} Q'}{(P \gg\!= Q) \xrightarrow{tr_1 \,@\, tr_2} Q'}$$

The *skip* process immediately terminates, returning (). If the left-hand side P of $\gg\!=$ can evolve to P' performing the events in tr, then the overall bind evolves similarly. If P can terminate after doing tr_1, returning x, and the continuation $Q(x)$ can evolve over tr_2 to Q' then the overall $\gg\!=$ can also evolve over the concatenation of tr_1 and tr_2, $tr_1 @ tr_2$, to Q'.

Often in CSP, one likes to show that there are no divergent states, a property called divergence freedom. It is captured by the following inductive-coinductive definition:

▶ **Definition 14** (Divergence Freedom).

$$\frac{-}{\checkmark_x \searrow \mathcal{R}} \qquad \frac{P \searrow \mathcal{R}}{\tau P \searrow \mathcal{R}} \qquad \frac{\mathrm{ran}(F) \subseteq \mathcal{R}}{\mathit{Vis}\, F \searrow \mathcal{R}} \qquad \mathit{div\text{-}free} \triangleq \bigcup \{\mathcal{R} \mid \mathcal{R} \subseteq \{P \mid P \searrow \mathcal{R}\}\}$$

Predicate $P \searrow \mathcal{R}$ is defined inductively. It requires that P stabilises to a *Ret*, or to a *Vis* whose coninuations are all contained in $\mathcal{R}$. Then, *div-free* is the largest set consisting of all sets $\mathcal{R} = \{P \mid P \searrow \mathcal{R}\}$, and is coinductively defined. If we can find an $\mathcal{R}$ such that for every $P \in \mathcal{R}$, it follows that $P \searrow \mathcal{R}$, that is $\mathcal{R}$ is closed under stabilisation, then any $P \in \mathcal{R}$ is divergence free. Essentially, $\mathcal{R}$ needs to enumerate the symbolic post-stable states of an ITree; for example $\mathcal{R} = \{\mathit{run}\, E\}$ satisfies the provisos and so $\mathit{run}\, E$ is divergence free. We have proved that $P \in \mathit{div\text{-}free} \Leftrightarrow (\nexists s \bullet P \xrightarrow{s} \mathit{div})$, which gives the operational meaning.

With our transition relation, we can define Roscoe's step relation, which is used to link the operational and denotational semantics of CSP [36, Section 9.5]. The utility of this definition, and the theorems that follow, is to permit symbolic verification of CSP processes by calculating their set-based characterisation.

▶ **Definition 15** (Roscoe's Step Relation).

$$(P \overset{s}{\Longrightarrow} P') \triangleq ((\exists\, t \in \Sigma\, \mathit{list} \bullet s = t \,@[\checkmark_x] \wedge P \xrightarrow{t} \checkmark_x \wedge P' = \mathit{stop}) \vee (\mathit{set}(s) \subseteq \Sigma \wedge P \xrightarrow{s} P'))$$

Here, $\mathit{set}(s)$ extracts the set of elements from a list. The step relation is similar to $\xrightarrow{s}$, except that the event type is adjoined with a special termination event ✓. We define the enlarged set $\Sigma^{\checkmark} \triangleq \Sigma \cup \{\checkmark_x \mid x \in \mathcal{S}\}$, which adds a family of events parametrised by return values, as in the semantics of Occam [34], which derives from CSP. A termination is signalled when the transition relation reaches a $\mathit{Ret}\, x$ in the ITree, in which case the trace is augmented with $\checkmark_x$ and the successor state is set to *stop*. We often use a condition of the form $\mathit{set}(s) \subseteq \Sigma$ to mean that no $\checkmark_x$ event is in s. We can now define the sets of traces, failures, and divergences [36]:

▶ **Definition 16** (Traces, Failures, and Divergences).

$$
\begin{aligned}
\mathit{traces}(P) &\triangleq \{s \mid \mathit{set}(s) \subseteq \Sigma^{\checkmark} \land (\exists\, P' \bullet P \overset{s}{\Rightarrow} P')\} \\
P \,\mathit{ref}\, E &\triangleq ((\exists\, F \bullet P = \mathit{Vis}\, F \land E \cap \mathrm{dom}(F) = \emptyset) \lor (\exists\, x \bullet P = \mathit{Ret}\, x \land \checkmark_x \notin E)) \\
\mathit{failures}(P) &\triangleq \left\{(s, X) \mid \mathit{set}(s) \subseteq \Sigma^{\checkmark} \land (\exists\, Q \bullet P \overset{s}{\Rightarrow} Q \land Q \,\mathit{ref}\, X)\right\} \\
\mathit{divergences}(P) &\triangleq \{s \mathbin{@} t \mid \mathit{set}(s) \subseteq \Sigma \land \mathit{set}(t) \subseteq \Sigma \land (\exists\, Q \bullet P \overset{s}{\Rightarrow} Q \land Q\Uparrow)\}
\end{aligned}
$$

The set $\mathit{traces}(P)$ is the set of all possible event sequences that P can perform. For $\mathit{failures}(P)$, we need to determine the set of events that an ITree is refusing, $P\,\mathit{ref}\,E$. If P is a visible event, $\mathit{Vis}\,F$, then any set of events E outside of $\mathrm{dom}(F)$ is refused. If P is a return event, $\mathit{Ret}\,x$, then every event other than $\checkmark_x$ is refused. With this, we can implement Roscoe's form for the failures. Finally, the divergences is simply a trace s leading to a divergent state $Q\Uparrow$, followed by any trace t. We exemplify these definitions with two calculations of failures:

$$
\begin{aligned}
\mathit{failures}(\mathit{inp}\; c\; A) = \;& \{([], E) \mid \forall\, x \in A \bullet c.x \notin E\} \cup \{([c.x], E) \mid x \in A \land \checkmark \notin E\} \\
& \cup \{([c.x, \checkmark_{()}], E) \mid x \in A\} \\
\mathit{failures}(P \gg= Q) = \;& \{(s, X) \mid \mathit{set}(s) \subseteq \Sigma \land (s, X \cup \{\checkmark_x \mid x \in \mathcal{S}\}) \in \mathit{failures}(P)\} \\
& \cup \{(s \mathbin{@} t, X) \mid \exists\, v \bullet s \mathbin{@} [\checkmark_v] \in \mathit{traces}(P) \land (t, X) \in \mathit{failures}(Q(v))\}
\end{aligned}
$$

The failures of $\mathit{inp}\; c\; A$ consists of (1) the empty trace, where no valid input on c is refused; (2) the trace where an input event $c.x$ occurred, and $\checkmark_{()}$ is not being refused; and (3) the trace where both $c.x$ and $\checkmark_{()}$ occurred, and every event is refused. The failures of $P \gg= Q$ consist of (1) the failures of P that do not reach a return, and (2) the terminating traces of P, ending in $\checkmark_v$ appended with a failure of $Q(v)$, the continuation. With the help of Isabelle's simplifier, these equations can be used to automatically calculate the failures and divergences, which can be easier to reason with than directly applying coinduction.

We conclude this section with some important properties of our semantic model:

▶ **Theorem 17** (Semantic Model Properties).

$$
\begin{aligned}
& (s, X) \in \mathit{failures}(P) \land (Y \cap \{x \mid s \mathbin{@} [x] \in \mathit{traces}(P)\} = \emptyset) \Rightarrow (s, X \cup Y) \in \mathit{failures}(P) \\
& s \in \mathit{divergences}(P) \land \mathit{set}(t) \subseteq \Sigma \Rightarrow s \mathbin{@} t \in \mathit{divergences}(P) \\
& P \approx Q \Rightarrow (\mathit{failures}(P) = \mathit{failures}(Q) \land \mathit{divergences}(P) = \mathit{divergences}(Q)) \\
& P \in \mathit{div\text{-}free} \Leftrightarrow \mathit{divergences}(P) = \emptyset \\
& P \in \mathit{div\text{-}free} \Rightarrow (\forall\, s\; a \bullet s \mathbin{@} [a] \in \mathit{traces}(P) \Rightarrow (s, \{a\}) \notin \mathit{failures}(P))
\end{aligned}
$$

The first two are standard healthiness conditions of the failures-divergences model [36], called ***F3*** and ***D1***, respectively. ***F3*** states that if (s, X) is a failure of P then any event that cannot subsequently occur after s, according to the traces, must also be refused. ***D1*** states that the set of divergences is extension closed. We have also proved that two weakly bisimilar processes have the same set of divergences and failures. The next result links the coinductive definition of divergence freedom and the set of divergences. The final result demonstrates that ITrees satisfy Roscoe's definition of determinism for CSP [36]: if an ITree P is divergence free then there is no trace after which an event can be both accepted and also refused.

5 Simulation by Code Generation

The Isabelle code generator [19, 18] can be used to extract code from (co)datatypes, functions, and other constructs, to functional languages like SML, Haskell, and Scala. Although ITrees can be infinite, this is not a problem for languages with lazy evaluation, and so we can step through the behaviour of an ITree. Code generation then allows us to support generation of verified simulators, and provides a potential route to correct implementations.

The main complexity is a computable representation of partial functions. Whilst $A \nrightarrow B$ is partly computable, all that we can do is apply it to a value and see whether it yields an output or not. For simulations and implementations, however, we typically want to determine a menu of enabled events for the user to select from. Moreover, calculation of a semantics for CSP operators like $\Box$ and $\|$ requires us to compute with partial functions. For this, we need a way of calculating values for functions dom, $\lhd$, and $\oplus$, which is not possible for arbitrary partial functions. Instead, we need a concrete implementation and a data refinement [18].

We choose associative lists as an implementation, $A \nrightarrow B \approx (A \times B)\,\mathsf{list}$, which limits us to finite constructions. However, it has the benefit of being easily pretty printed and so makes the simulator easier to implement. More sophisticated implementations are possible, as the core theory of ITrees is separated from the code generation setup. To allow us to represent partial functions by associative lists, we need to define a mapping function:

```
fun pfun_alist :: "('a × 'b) list ⇒ ('a ⇸ 'b)" where
"pfun_alist [] = {↦}" | "pfun_alist ((k,v) # f) = pfun_alist f ⊕ {k ↦ v}"
```

This recursive function converts an associative list to a partial function, by adding each pair in the list as a maplet. We generally assume that associative lists preserve distinctness of keys. However, for this function, keys which occur earlier take priority. With this function we can then demonstrate how the different partial function operators can be computed. We prove the following congruence equations as theorems in Isabelle/HOL.

$$
\begin{aligned}
(\mathit{pfun\_alist}\, f) \oplus (\mathit{pfun\_alist}\, g) &= \mathit{pfun\_alist}\,(g \mathbin{@} f) \\
A \lhd (\mathit{pfun\_alist}\, f) &= \mathit{pfun\_alist}\,(\mathit{AList.restrict}\, A\, m) \\
(\lambda\, x \in (\mathit{set}\, xs) \bullet f(x)) &= \mathit{pfun\_alist}\,(\mathit{map}\,(\lambda\, k \bullet (k, f\, k))\; xs)
\end{aligned}
$$

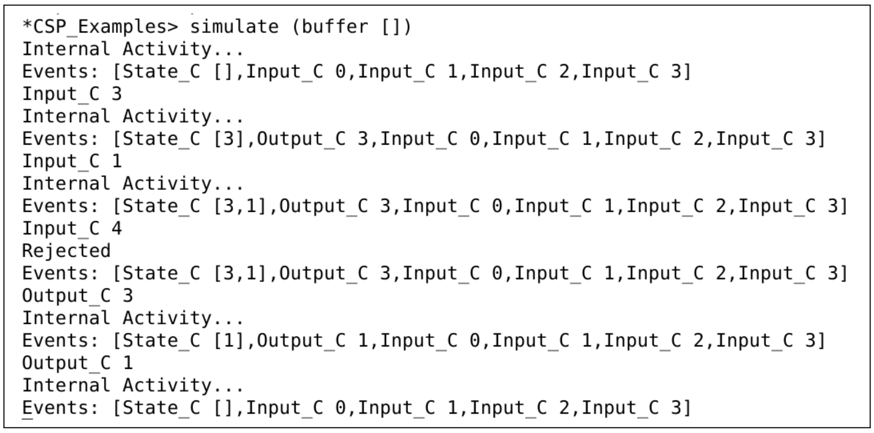
Override ($\oplus$) is expressed by concatenating the associative lists in reverse order. Domain restriction ($\lhd$) has an efficient implementation in Isabelle, *AList.restrict*, which we use. For a partial λ-abstraction, we assume that the domain set is characterised by a list $(\mathit{set}\, xs)$. Then, a λ term can be computed by mapping the body function f over xs.

With these equations, we can set up the code generator. The idea is to designate certain representations of abstract types as code datatypes in the target language, of which each mapping function is a constructor. For sets, the following Haskell code datatype is produced:

```
data Set a = Set [a] | Coset [a] deriving (Read, Show);
```

A set is represented as a list of values using the constructor `Set`, which corresponds to the function *set*. It is often the case that we wish to capture a complement of another set, and so there is also the constructor `Coset` for a set whose elements are all those not in the given list. Functions on sets are then computed by code equations, which provide the implementation for each concrete representation. The membership function *member* is implemented like this:

```
member :: forall a. (Eq a) => a -> Set a -> Bool;
member x (Coset xs) = not (x `elem` xs); member x (Set xs) = xs `elem` x;
```



```
*CSP_Examples> simulate (buffer [])
Internal Activity...
Events: [State_C [],Input_C 0,Input_C 1,Input_C 2,Input_C 3]
Input_C 3
Internal Activity...
Events: [State_C [3],Output_C 3,Input_C 0,Input_C 1,Input_C 2,Input_C 3]
Input_C 1
Internal Activity...
Events: [State_C [3,1],Output_C 3,Input_C 0,Input_C 1,Input_C 2,Input_C 3]
Input_C 4
Rejected
Events: [State_C [3,1],Output_C 3,Input_C 0,Input_C 1,Input_C 2,Input_C 3]
Output_C 3
Internal Activity...
Events: [State_C [1],Output_C 1,Input_C 0,Input_C 1,Input_C 2,Input_C 3]
Output_C 1
Internal Activity...
Events: [State_C [],Input_C 0,Input_C 1,Input_C 2,Input_C 3]
```

Figure 1 Simulating the CSP buffer in the Glasgow Haskell Interpreter.

Each case for the function corresponds to a code equation. The function **elem** is the Haskell prelude function that checks whether a value is in a list. This kind of representation ensures correctness of the generated code with respect to the Isabelle specifications. Similarly to sets, we can code generate the following representation for partial functions:

```
data Pfun a b = Pfun_alist [(a, b)];

dom :: forall a b. Pfun a b -> Set a;
dom (Pfun_alist xs) = Set (map fst xs);
```

A partial function has a single constructor, although it is possible to augment this with additional representations. Each code equation likewise becomes a case for the corresponding recursive function, as illustrated by the domain function. Finally, we can code generate interaction trees, which are represented by a very compact datatype:

```
data Itree a b = Ret b | Sil (Itree a b) | Vis (Pfun a (Itree a b));
```

Each semantic definition, including corecursive functions, are also automatically mapped to Haskell functions. We illustrate the code generated for external choice below:

```
extchoice :: (Eq a, Eq b) => Itree a b -> Itree a b -> Itree a b;
extchoice p q = (case (p, q) of {
    (Ret r, Ret y) -> (if r == y then Ret r else Vis zero_pfun);
    (Ret _, Sil qa) -> Sil (extchoice p qa); (Ret r, Vis _) -> Ret r;
    (Sil pa, _) -> Sil (extchoice pa q); (Vis _, Ret a) -> Ret a;
    (Vis _, Sil qa) -> Sil (extchoice p qa);
    (Vis f, Vis g) -> Vis (map_prod f g); });
```

The map_prod function corresponds to $\odot$, and is defined in terms of the corresponding code generated functions for partial functions. The external choice operator ($\Box$) is simply defined as an infinitely recursive function with each of the corresponding cases in Definition 7.

For constructs like *inp* (Definition 6), there is more work to support code generation, since these can potentially produce an infinite number of events which cannot be captured by an associative list. Consider, for example, *inp* $c\,\{0..\}$, for $c : \mathbb{N} \overset{\triangle}{\longrightarrow} E$, which can produce any event $c.i$ for $i \geq 0$. We can code generate this by limiting the value set to be finite, for example $\{0..3\}$. Then, the code generator maps this to a list $[0, 1, 2, 3]$, which is computable. Thus, we can finally export code for concrete examples using the operator implementations.

We can now implement a simple simulator, the code for which is shown below:

```
sim_cnt :: (Eq e, Show e, Read e, Show s) => Int -> Itree e s -> IO ();
sim_cnt n (Ret x) = putStrLn ("Terminated:␣" ++ show x);
sim_cnt n (Sil p) =
  do { if (n == 0) then putStrLn "Internal␣Activity..." else return ();
       if (n >= 20)
       then do { putStr "Many␣steps␣(>␣20);␣Continue?"; q <- getLine;
                 if (q=="Y") then sim_cnt 0 p else putStrLn "Ended."; }
       else sim_cnt (n + 1) p };
sim_cnt n (Vis (Pfun_alist [])) = putStrLn "Deadlocked.";
sim_cnt n t@(Vis (Pfun_alist m)) =
  do { putStrLn ("Events:␣" ++ show (map fst m)); e <- getLine;
       case (reads e) of
         []       -> do { putStrLn "No␣parse"; sim_cnt n t }
         [(v, _)] -> case (lookup v m) of
                       Nothing -> do { putStrLn "Rejected"; sim_cnt n t }
                       Just k -> sim_cnt 0 k };
simulate = sim_cnt 0;
```

The idea is to step through τs until we reach either a $\checkmark_x$, in which case we terminate, or a *Vis*, in which we case the user can choose an option. Since divergence is a possibility, we limit the number of τs that the will be skipped. After 20 τ steps, the user can choose to continue or abort the simulation. If an empty event choice is encountered, then the simulation terminates due to deadlock. Otherwise, it displays a menu of events, allows the user to choose one, and then recurses following the given continuation. The simulator currently depends on associative lists to represent choices, but other implementations are possible.

In order to apply the simulator, we need only augment the generated code for a particular ITree with the simulator code. Figure 1 shows a simulation of the CSP buffer in §3, with the possible inputs limited to $\{0..3\}$. We provide an empty list as a parameter for the initial state. The simulator tells us the events enabled, and allows us to pick one. If we try and pick a value not enabled, the simulator rejects this. Since lenses and expressions can also be code generated, we can also simulate the *Circus* version of the buffer, with the same output.

As a more sophisticated example, we have implemented a distributed ring buffer, which is adopted from the original *Circus* paper [42]. The idea is to represent a buffer as a ring of one-place cells, and a controller that manages the ring. It has the following form:

$$(\mathit{Controller} \mathbin{[\![} \{\mathit{rd}.c, \mathit{wrt}.c \mid c \in \mathbb{N}\} \mathbin{]\!]} (\mathbin{|||}\, i \in \{0..\mathit{maxbuff}\} \bullet \mathit{Cell}(i))) \setminus \{\mathit{rd}.c, \mathit{wrt}.c \mid c \in \mathbb{N}\}$$

where $\mathit{rd}.c$ and $\mathit{wrt}.c$ are internal channels for the controller to communicate with the ring. Each cell is a single place buffer with a state variable val, and has the form

$$\mathit{Cell}(i) \triangleq \mathit{wrt}?c \rightarrow \mathit{val} := v \mathbin{⨾} \mathit{loop}(\mathit{wrt}?c \rightarrow \mathit{val} := v \mathbin{\Box} \mathit{rd}!\mathit{val} \rightarrow \textsf{Skip})$$

The cells are arranged through indexed interleaving, and $\mathit{maxbuff}$ is the buffer size. The channels *Input* and *Output* are used for communicating with the overall buffer. Space will not permit further details. The simulator can efficiently simulate this example, for a small ring with 5 cells, with a similar output to Figure 1, which is a satisfying result.

We were also able to simulate the ring buffer with 100 cells, which requires about 3 seconds to compute the next step. With 1000 cells, the simulator takes more than a minute to calculate the next transition. The highest number of cells we could reasonably simulate is around 250. However, we have made no attempt to optimise the code, and several data types could be replaced with efficient implementations to improve scalability. Thus, as an approach to simulation and potentially implementation, this is very promising.

6 Related Work

Infinite trees are a ubiquitous model for concurrency [40]. In particular, ITrees can be seen as a restricted encoding of Milner's synchronisation trees [27, 41, 28]. In contrast to ITrees, synchronisation trees allow multiple events from each node, including both visible and τ events. They have seen several generalisations, most recently by Ferlez et al. [10], who formalise Generalized Synchronisation Trees based on partial orders, define bisimulation relations [11], and apply them to hybrid systems. Our work is different, because ITrees use explicit coinduction and corecursion, but there are likely mutual insights to be gained.

ITrees naturally support deterministic interactions, which makes them ideal for implementations. Milner extensively discusses determinism in [28, chapter 11], a property which is imposed by construction in our operators. Similarly, Hoare defines a deterministic choice operator $a \to P \mid b \to Q$ in [21, page 29], which is similar to ours except that Hoare's operator imposes determinism syntactically, where we introduce deadlock.

ITrees [43], and their mechanisation in Coq, have been applied in various projects as a way of defining abstract yet executable semantics [23, 44, 26, 45, 46, 25, 37]. They have been used to verify C programs [23] and a HTTP key-value server [25]. The Coq mechanisation uses features not available in Isabelle, notably type constructor variables. Specifically, in [43] the *Vis* constructor has two parameters, rather than one, for the enabled events (i.e. channels) $e : \mathcal{E}\,\mathcal{A}$ and $k : \mathcal{A} \to \mathit{itree}\ \mathcal{E}\ \mathcal{R}$, a total function, for the continuation. There, $\mathcal{E}$ is a type constructor over $\mathcal{A}$, the type of data. Our work avoids this, with no apparent loss of generality, by fixing an event universe, E; using partial functions to represent visible event choices; and using prisms [33] to characterise channels. We can encode the two parameter $\mathit{Vis}\ e\ k$ as $\Box\, x \in \mathrm{dom}(\mathit{match}_e) \to k(\mathit{match}_e(x))$ with $e : A \xrightarrow{\triangle} E$. The benefit of having a fixed E is that ITrees become much simpler semantic objects. Traces can be represented as lists, rather than the bespoke type used in [43]. These are amenable to first-order automated proof [5], which has allowed us to develop our library quickly and with minimal effort.

7 Conclusions

In this paper we showed how Interaction Trees [43] can be used to develop verified simulations for state-rich process languages with the help of Isabelle codatatypes [4] and the code generator [19, 18]. Our early results indicate that the technique provides both tractable verification, with the help of Isabelle's proof automation [5] and efficient simulation. We applied our technique to the CSP and *Circus* process languages, though it is applicable to a variety of other process algebraic languages.

So far, we have focused primarily on deterministic processes, since these are easier to implement. This is not, however, a limitation of the approach. There are at least three approaches that we will investigate to handling nondeterminism in the future: (1) use of a dedicated indexed nondeterminism event; (2) extension of ITrees to permit a computable set of events following a τ; (3) a further Kleisli lifting of ITrees into sets. Moreover, we will formally link ITrees to our formalisation of reactive contracts [15, 16], which provides both a denotational semantics for *Circus* and a refinement calculus for reactive systems, building on our link with failures-divergences. We will implement the remaining CSP operators, such as renaming and interruption. We will also further investigate the failures-divergence semantics of our ITree process operators, and determine whether failures-divergences equivalence entails weak bisimulation. Finally we will provide a more user friendly interface for our simulator as found in animators like FDR4's probe tool [17] and ProB [24] for Event-B.

Our work has many practical applications in production of verified simulations. We intend to use it to mechanise a semantics for the RoboChart [29] and RoboSim [9] languages, which are formal UML-like languages for modelling robots with denotational semantics based in CSP. This will require us to consider discrete time, which we believe can be supported using a dedicated time event in ITrees, similar to tock-CSP [35]. This will build on our colleagues' work with ✓*-tock* [2], a new semantics for tock-CSP. This will open up a pathway from graphical models to verified implementations of autonomous robotic controllers. In concert with this, we will also explore links to our other theories for hybrid systems [31, 13], to allow verification of controllers in the presence of a continuously evolving environment.

References

1 R.-J. Back and J. Wright. *Refinement Calculus: A Systematic Introduction.* Springer, 1998.

2 J. Baxter, P. Ribeiro, and A. Cavalcanti. Sound reasoning in tock-CSP. *Acta Informatica*, April 2021. doi:10.1007/s00236-020-00394-3.

3 J. C. Blanchette, A. Bouzy, A. Lochbihler, A. Popescu, and D. Traytel. Friends with Benefits: Implementing Corecursion in Foundational Proof Assistants. In *Programming Languages and Systems, 26th European Symposium on Programming (ESOP)*, 2017.

4 J. C. Blanchette, J. Hölzl, A. Lochbihler, L. Panny, A. Popescu, and D. Traytel. Truly modular (co)datatypes for Isabelle/HOL. In Gerwin Klein and Ruben Gamboa, editors, *5th Intl. Conf. on Interactive Theorem Proving (ITP)*, volume 8558 of *LNCS*, pages 93–110. Springer, 2014.

5 J. C. Blanchette, C. Kaliszyk, L. C. Paulson, and J. Urban. Hammering towards QED. *Journal of Formalized Reasoning*, 9(1), 2016. doi:10.6092/issn.1972-5787/4593.

6 J. C. Blanchette, A. Popescu, and D. Traytel. Foundational extensible corecursion: a proof assistant perspective. In *20th Intl. Conf. on Functional Programming (ICFP)*, pages 192–204. ACM, August 2015. doi:10.1145/2858949.2784732.

7 J. C. Blanchette, A. Popescu, and D. Traytel. Soundness and completeness proofs by coinductive methods. *Journal of Automated Reasoning*, 58:149–179, 2017. doi:10.1007/s10817-016-9391-3.

8 S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM*, 31(3):560–599, 1984. doi:10.1145/828.833.

9 A. Cavalcanti, A. Sampaio, A. Miyazawa, P. Ribeiro, M. Filho, A. Didier, W. Li, and J. Timmis. Verified simulation for robotics. *Science of Computer Programming*, 174:1–37, 2019. doi:10.1016/j.scico.2019.01.004.

10 J. Ferlez, R. Cleaveland, and S. Marcus. Generalized synchronization trees. In *Proc. 17th Intl. Conf. on Foundations of Software Science and Computation Structures (FOSSACS)*, volume 8412 of *LNCS*, pages 304–319. Springer, 2014. doi:10.1007/978-3-642-54830-7_20.

11 J. Ferlez, R. Cleaveland, and S. I. Marcus. Bisimulation in behavioral dynamical systems and generalized synchronization trees. In *Proc. 2018 IEEE Conf. on Decision and Control (CDC)*, pages 751–758. IEEE, 2018. doi:10.1109/CDC.2018.8619607.

12 J. Foster. *Bidirectional programming languages.* PhD thesis, University of Pennsylvania, 2009.

13 S. Foster. Hybrid relations in Isabelle/UTP. In *7th Intl. Symp. on Unifying Theories of Programming (UTP)*, volume 11885 of *LNCS*, pages 130–153. Springer, 2019.

14 S. Foster, J. Baxter, A. Cavalcanti, J. Woodcock, and F. Zeyda. Unifying semantic foundations for automated verification tools in Isabelle/UTP. *Science of Computer Programming*, 197, October 2020. doi:10.1016/j.scico.2020.102510.

15 S. Foster, A. Cavalcanti, S. Canham, J. Woodcock, and F. Zeyda. Unifying theories of reactive design contracts. *Theoretical Computer Science*, 802:105–140, January 2020. doi:10.1016/j.tcs.2019.09.017.

16 S. Foster, K. Ye, A. Cavalcanti, and J. Woodcock. Automated verification of reactive and concurrent programs by calculation. *Journal of Logical and Algebraic Methods in Programming*, 121, June 2021. doi:10.1016/j.jlamp.2021.100681.

17 T. Gibson-Robinson, P. Armstrong, A. Boulgakov, and A. W. Roscoe. FDR3 — A Modern Refinement Checker for CSP. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 8413 of *LNCS*, pages 187–201, 2014.

18 F. Haftmann, A. Krauss, O. Kuncar, and T. Nipkow. Data refinement in Isabelle/HOL. In *Proc. 4th Intl. Conf. on Interactive Theorem Proving (ITP)*, volume 7998 of *LNCS*, pages 100–115. Springer, 2013.

19 F. Haftmann and T. Nipkow. Code generation via higher-order rewrite systems. In *10th Intl. Symp. on Functional and Logic Programming (FLOPS)*, volume 6009 of *LNCS*, pages 103–117. Springer, 2010.

20 Matthew Hennessy and Tim Regan. A process algebra for timed systems. *Information and Computation*, 117(2):221–239, 1995.

21 C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

22 C. A. R. Hoare, I. Hayes, J. He, C. Morgan, A. Roscoe, J. Sanders, I. Sørensen, J. Spivey, and B. Sufrin. The laws of programming. *Communications of the ACM*, 30(8):672–687, August 1987.

23 Nicolas Koh, Yao Li, Yishuai Li, Li yao Xia, Lennart Beringer, Wolf Honoré, William Mansky, Benjamin C. Pierce, and Steve Zdancewic. From C to Interaction Trees: Specifying, Verifying, and Testing a Networked Server. In *Proc. 8th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*, 2019. doi:10.1145/3293880.3294106.

24 M. Leuschel and M. Butler. ProB: an automated analysis toolset for the B method. *Int J Softw Tools Technol Transf*, 10:185–203, 2008. doi:10.1007/s10009-007-0063-9.

25 Yishuai Li, Benjamin C. Pierce, and Steve Zdancewic. Model-based testing of networked applications. In *Proc. 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*, 2021.

26 William Mansky, Wolf Honoré, and Andrew W. Appel. Connecting higher-order separation logic to a first-order outside world. In *Proc. 29th European Symposium on Programming (ESOP)*, 2020.

27 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

28 Robin Milner. *Communication and Concurrency.* Prentice Hall, 1989.

29 A. Miyazawa, P. Ribeiro, W. Li, A. Cavalcanti, J. Timmis, and J. Woodcock. RoboChart: modelling and verification of the functional behaviour of robotic applications. *Software and Systems Modelling*, January 2019. doi:10.1007/s10270-018-00710-z.

30 C. Morgan. *Programming from Specifications.* Prentice-Hall, January 1996.

31 J. H. Y. Munive, G. Struth, and S. Foster. Differential Hoare logics and refinement calculi for hybrid systems with Isabelle/HOL. In *RAMiCS*, volume 12062 of *LNCS*. Springer, April 2020. doi:10.1007/978-3-030-43520-2_11.

32 M. Oliveira, A. Cavalcanti, and J. Woodcock. A UTP semantics for Circus. *Formal Aspects of Computing*, 21:3–32, 2009. doi:10.1007/s00165-007-0052-5.

33 M. Pickering, J. Gibbons, and N. Wu. Profunctor optics: Modular data accessors. *The Art, Science, and Engineering of Programming*, 1(2), 2017. doi:10.22152/programming-journal.org/2017/1/7.

34 A. W. Roscoe. Denotational semantics for Occam. In *Intl. Seminar on Concurrency*, volume 197 of *LNCS*, pages 306–329. Springer, 1984.

35 A. W. Roscoe. *The Theory and Practice of Concurrency.* Prentice-Hall, 2005.

36 A. W. Roscoe. *Understanding Concurrent Systems.* Springer, 2010.

37 Lucas Silver and Steve Zdancewic. Dijkstra monads forever: Termination-sensitive specifications for Interaction Trees. *Proceedings of the ACM on Programming Languages*, 5(POPL), January 2021. doi:10.1145/3434307.

38 M. Spivey. *The Z-Notation - A Reference Manual.* Prentice Hall, Englewood Cliffs, N. J., 1989.

39 S. Taha, B. Wolff, and L. Ye. Philosophers may dine – definitively! In *Proc. 16th Intl. Conf. on Integrated Formal Methods*, LNCS. Springer, 2020. doi:10.1007/978-3-030-63461-2_23.

40 R. J. van Glabbeek. Notes on the methodology of CCS and CSP. *Theoretical Computer Science*, 1997.

41 G. Winskel. Synchronisation trees. *Theoretical Computer Science*, 34(1-2):33–82, 1984.

42 J. Woodcock and A. Cavalcanti. A concurrent language for refinement. In A. Butterfield, G. Strong, and C. Pahl, editors, *Proc. 5th Irish Workshop on Formal Methods (IWFM)*, Workshops in Computing. BCS, July 2001.

43 L.-Y. Xia, Y. Zakowski, P. He, C.-K. Hur, G. Malecha, B. C. Pierce, and S. Zdancewic. Interaction trees: Representing recursive and impure programs in Coq. In *Proc. 47th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*. ACM, 2020. doi:10.1145/3371119.

44 Yannick Zakowski, Paul He, Chung-Kil Hur, and Steve Zdancewic. An equational theory for weak bisimulation via generalized parameterized coinduction. In *Proc. 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP)*, 2020. doi:10.1145/3372885.3373813.

45 Vadim Zaliva, Ilia Zaichuk, and Franz Franchetti. Verified translation between purely functional and imperative domain specific languages in HELIX. In *Proc. 12th International Conference on Verified Software: Theories, Tools, Experiments (VSTTE)*, 2020.

46 Hengchu Zhang, Wolf Honoré, Nicolas Koh, Yao Li, Yishuai Li, Li-Yao Xia, Lennart Beringer, William Mansky, Benjamin C. Pierce, and Steve Zdancewic. Verifying an HTTP key-value server with Interaction Trees and VST. In *Proc. 12th International Conference on Interactive Theorem Proving (ITP)*, 2021. doi:10.4230/LIPIcs.ITP.2021.32.

Fibrational Initial Algebra-Final Coalgebra Coincidence over Initial Algebras: Turning Verification Witnesses Upside Down

Mayuko Kori
The Graduate University for Advanced Studies (SOKENDAI), Hayama, Japan
National Institute of Informatics, Tokyo, Japan

Ichiro Hasuo
The Graduate University for Advanced Studies (SOKENDAI), Hayama, Japan
National Institute of Informatics, Tokyo, Japan

Shin-ya Katsumata
National Institute of Informatics, Tokyo, Japan

Abstract

The coincidence between initial algebras (IAs) and final coalgebras (FCs) is a phenomenon that underpins various important results in theoretical computer science. In this paper, we identify a general fibrational condition for the IA-FC coincidence, namely in the fiber over an initial algebra in the base category. Identifying (co)algebras in a fiber as (co)inductive predicates, our fibrational IA-FC coincidence allows one to use *coinductive* witnesses (such as invariants) for verifying *inductive* properties (such as liveness). Our general fibrational theory features the technical condition of stability of chain colimits; we extend the framework to the presence of a monadic effect, too, restricting to fibrations of complete lattice-valued predicates. Practical benefits of our categorical theory are exemplified by new "upside-down" witness notions for three verification problems: probabilistic liveness, and acceptance and model-checking with respect to bottom-up tree automata.

2012 ACM Subject Classification Theory of computation → Categorical semantics

Keywords and phrases initial algebra, final coalgebra, fibration, category theory

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.21

Related Version *Extended Version*: `https://arxiv.org/abs/2105.04817`

Funding *Ichiro Hasuo*: ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST
Shin-ya Katsumata: ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST

1 Introduction

Categorical Algebras and Coalgebras. Categorical algebras and coalgebras are omnipresent in theoretical computer science. For a category $\mathbb{C}$ and an endofunctor $F\colon \mathbb{C} \to \mathbb{C}$, an F-algebra is a $\mathbb{C}$-morphism $a\colon FX \to X$, while an F-coalgebra is $c\colon X \to FX$. These structures occur in many different settings with different $\mathbb{C}$ and F; the identification of such (co)algebras has yielded a number of concrete benefits, such as rigorous system/program semantics, verification methods, and programming language constructs.

One principal use of categorical (co)algebras is as models of *data structures* such as terms and state-based systems. Examples include modeling of inductive datatypes by initial algebras [6], and the theory of coalgebras [18, 26] that captures state-based behaviors. Here, the base category $\mathbb{C}$ is typically that of (structured) sets and (structure-preserving) maps. (In this paper, such a category will constitute a *base category* of a fibration).

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 21; pp. 21:1–21:22
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Another principal use of (co)algebras is as logical *recursive specifications.* Here the base category $\mathbb{C}$ is typically a complete lattice of truth values (such as $\mathbf{2} = \{\bot, \top\}$) and the functor $F\colon \mathbb{C} \to \mathbb{C}$ is identified with a monotone function. Liveness properties are modeled by least fixed points (lfp's); safety properties are greatest ones (gfp's); and by the classic Knaster–Tarski theorem, these are identified with initial algebras and final coalgebras, respectively. (In this paper, such a category will appear as a *fibre category* of a fibration).

Initial Algebras and Final Coalgebras. In the above variety of occurrences of (co)algebras, *initial algebras* and *final coalgebras* play key roles. Their definition is by suitable universality: $\beta\colon F(\mu F) \to \mu F$ is initial if there is a unique algebra morphism from β to an arbitrary algebra $a\colon FX \to X$; and dually for final coalgebras.

$$\begin{tikzcd}
F(\mu F) \arrow[r, dotted, tail] \arrow[d, "\text{init.}"', "\cong"] & FX \arrow[d] & FX \arrow[r, dotted, tail] & F(\nu F) \\
\mu F \arrow[r, dotted, tail] & X & X \arrow[u] \arrow[r, dotted, tail] & \nu F \arrow[u, "\cong"', "\text{final}"]
\end{tikzcd}$$

Their (co)algebra structures are isomorphisms by the Lambek lemma. The latter extends the Knaster–Tarski theorem from lattices to categories.

In many occurrences of (co)algebras in computer science, initial algebras represent *finitary* entities while final coalgebras represent *infinitary* entities. For example, when $\mathbb{C} = \mathbf{Set}$ (the category of sets and functions) and F is a functor that models a datatype constructor, the carrier μF of an initial algebra represents the *inductive datatype* – collecting all finite trees "of shape F" – while the carrier νF of a final coalgebra is for the *coinductive datatype* and collects all (finite and infinite) trees. This intuition is found also in the logical (co)algebras: liveness properties (initial algebras) can be witnessed within finitely many steps, while safety properties (final coalgebras) are verified only after infinitely many steps.

Initial Algebra-Final Coalgebra Coincidence. In this paper, we are interested in the coincidence of an initial algebra and a final coalgebra (the *IA-FC coincidence*). While it may sound unlikely in view of the contrast between finitary and infinitary, the coincidence has been found in different areas in computer science, underpinning fundamental results.

One example is in *domain theory*: cpo-enrichment yields the IA-FC coincidence, which is used to solve recursive domain equations of mixed variance [11, 13, 27, 32]. Another example is in *process semantics*: specifically, in the coalgebraic characterization of finite trace semantics [15], the IA-FC coincidence in some Kleisli categories $\mathbf{Kl}(T)$ has been observed.

Contribution: the Fibrational IF/I Coincidence and Application to Verification Witnesses. In this paper, we identify a general *fibrational* condition for the IA-FC coincidence: under mild assumptions, we have the IA-FC coincidence in the fiber over an initial algebra in the base category (the *IF/I coincidence*). Identifying the base IA as a datatype, and the fibre IA/FC as lfp/gfp specifications, the IF/I coincidence implies the coincidence between *induction* and *coinduction* as reasoning principles, assuming they are over a (finitary) algebraic datatype.

This coincidence allows us to *turn witness notions upside down*, that is, to use coinductive witness notions for establishing inductive properties. In general, inductive witness notions for lfp properties (such as ranking functions) tend to be more complex than coinductive witness notions for gfp properties (such as invariants). When we have the IF/I coincidence, the latter can now be used for lfp properties.

Our technical contributions are as follows. We work with a *fibration* $p\colon \mathbb{E} \to \mathbb{B}$, where $\mathbb{B}$ is intuitively a category of sets and functions, and $\mathbb{E}$ equips these sets with predicates.

- We identify a general fibrational framework for what we call the *IF/I coincidence* – the coincidence of IAs (lfp predicates) and FCs (gfp predicates) in the fiber over an initial algebra in $\mathbb{B}$ (an inductive datatype). The IF/I coincidence relies only on mild fibrational assumptions, notable among which are *fibredness* of functors and *stability* of certain colimits. Although we restrict fibrations to posetal ones in the main text (§4), a similar result for general fibrations can be shown (Appendix A).
- As a notable class of examples, in §5 we show that the fibration of Ω-valued predicates exhibits the IF/I coincidence (where Ω is an arbitrary complete lattice for truth values). Furthermore, we study the IF/I coincidence in the presence of monadic effects [23], building on the fibrational framework from [4].
- These theoretical results are used to obtain coinductive (invariant-like) witness notions for inductive (lfp, liveness) properties. Specifically, we present new witness notions for probabilistic verification (§6) and verification with tree automata as specifications (§7).

Related Work. Many works are discussed in the technical sections; we discuss some others.

The work [25] shows uniqueness of fixed points above what is called a *minimal invariant*; the latter corresponds to the lifting of a morphism which is both an initial algebra and a final coalgebra. Our IF/I coincidence can yield such lifting under some assumptions (see Thm. 3.6). The proof in [25] relies on homset enrichment, unlike our fibrational framework.

One of our main ideas is to use the IA-FC coincidence for novel proof methods for recursive specifications (§6–7), mixing lfp's and gfp's. This is pursued also in [8, 31] where *corecursive algebras* induce the lfp-gfp coincidence.

Organization. After recalling fibrations and the chain construction of initial algebras in §2, we formulate our IF/I coincidence in §3 and present sufficient conditions for the coincidence in §4. In §5, these results are specialized to fibrations of Ω-valued predicates, where we additionally include monadic effects. This paves the way to the concrete applications in §6–7, where we present seemingly new verification techniques for probabilistic liveness and witnesses of tree automata. We defer many proofs to the appendix.

2 Preliminaries

2.1 Fibrations

A fibration $p : \mathbb{E} \to \mathbb{B}$ is a functor that models indexing and substitution. That is, a functor $p : \mathbb{E} \to \mathbb{B}$ can be seen as a family of categories $(\mathbb{E}_X)_{X \in \mathbb{B}}$ that is equipped with substitution functors that change the index X.

In our examples, the base category $\mathbb{B}$ is that of sets and (potentially effectful) functions; and the total category $\mathbb{E}$ models "predicates" over sets in $\mathbb{B}$. We review a minimal set of definitions and results on fibrations. See [19] for details.

▶ **Definition 2.1** (fibre, fibration)**.** Let $p : \mathbb{E} \to \mathbb{B}$ be a functor.

For each $X \in \mathbb{B}$, the *fibre category* (or simply *fibre*) $\mathbb{E}_X$ over X is the category with objects $P \in \mathbb{E}$ such that $pP = X$ and morphisms $f : P \to Q$ such that $pf = \mathrm{id}_X$. An object $P \in \mathbb{E}_X$ is said to be *above* X and a morphism $f \in \mathbb{E}_X$ is said to be *vertical*.

A morphism $f : P \to Q$ in $\mathbb{E}$ is *cartesian* if it satisfies the following universality: for each $g : R \to Q$ in $\mathbb{E}$ and $k : pR \to pP$ in $\mathbb{B}$ with $pg = pf \circ k$, there exists a unique morphism $h : R \to P$ satisfying $g = f \circ h$ and $ph = k$. See the diagram below.

$$\begin{tikzcd}
\mathbb{E} \arrow[ddd, "p"'] & R \arrow[drr, bend left, "g"] \arrow[dr, dotted, "h"'] & & & & \\
& & P \arrow[r, "f"'] & Q & l^*Q \arrow[r, "\bar{l}"] & Q \\
& pR \arrow[drr, bend left, "pg"] \arrow[dr, "k"'] & & & & \\
\mathbb{B} & & pP \arrow[r, "pf"'] & pQ & X \arrow[r, "l"'] & pQ
\end{tikzcd}$$

The functor $p : \mathbb{E} \to \mathbb{B}$ is a *fibration* if, for each $Q \in \mathbb{E}$ and each $l : X \to pQ$ in $\mathbb{B}$, there exists $l^*Q \in \mathbb{E}$ and a morphism $\bar{l} : l^*Q \to Q$ such that $p\bar{l} = l$ and $\bar{l}$ is cartesian.

The functor $p : \mathbb{E} \to \mathbb{B}$ is an *opfibration* if $p^{\mathrm{op}} : \mathbb{E}^{\mathrm{op}} \to \mathbb{B}^{\mathrm{op}}$ is a fibration. A functor that is both a fibration and an opfibration is called a *bifibration.*

When p is a fibration, the correspondence from Q to l^*Q described above induces the *substitution functor* $l^* : \mathbb{E}_Y \to \mathbb{E}_X$ which replaces the index. The following characterization of bifibrations is useful for us: a fibration p is a bifibration if and only if each substitution functor $l^* : \mathbb{E}_Y \to \mathbb{E}_X$ (often called a *pullback*) has a left adjoint $l_* : \mathbb{E}_X \to \mathbb{E}_Y$ (often called a *pushforward*).

We are interested in reasoning over algebraic datatypes, that is in categorical terms, predicates in $\mathbb{E}_{\mu F}$ over the carrier μF of the initial algebra for $F \colon \mathbb{B} \to \mathbb{B}$. For this purpose we often consider a tuple $(p, F, \dot{F})$ in the following definition.

▶ **Definition 2.2** ((fibred) lifting). Let $p : \mathbb{E} \to \mathbb{B}$ be a functor and F be an endofunctor on $\mathbb{B}$. We say that an endofunctor $\dot{F}$ on $\mathbb{E}$ is a *lifting* of F along p if $p \circ \dot{F} = F \circ p$ (see above).

Assuming that p is a fibration, a lifting $\dot{F}$ is *fibred* if $\dot{F}$ preserves cartesian morphisms.

In this paper, we focus on a certain class of posetal fibrations called $\mathbf{CLat}_\wedge$-fibrations. They can be seen as *topological functors* [17] whose fibres are posets. This class abstracts treatment of spacial and logical structures.

▶ **Definition 2.3** ($\mathbf{CLat}_\wedge$-fibration). A $\mathbf{CLat}_\wedge$*-fibration* is a fibration $p \colon \mathbb{E} \to \mathbb{B}$ where each fibre $\mathbb{E}_X$ is a complete lattice and each substitution $f^* \colon \mathbb{E}_Y \to \mathbb{E}_X$ preserves all meets $\bigwedge$.

In each fibre $\mathbb{E}_X$, the order is denoted by $\leq_X$ or $\leq$. Its least and greatest elements are denoted by $\bot_X$ and $\top_X$; its join and meet are denoted by $\bigvee$ and $\bigwedge$.

The above simple axioms of $\mathbf{CLat}_\wedge$-fibrations induce many useful structures [20, 28]. One of them is that a $\mathbf{CLat}_\wedge$-fibration is always a bifibration whose pushforwards f_* arise essentially by Freyd's adjoint functor theorem. Another one is that $\mathbf{CLat}_\wedge$-fibrations lift colimits. This is proved by [19, Prop. 9.2.2 and Exercise 9.2.4].

▶ **Proposition 2.4.** *Let $p : \mathbb{E} \to \mathbb{B}$ be a $\mathbf{CLat}_\wedge$-fibration.*

1. *p is a bifibration.*
2. *If $\mathbb{B}$ is (co)complete then $\mathbb{E}$ is also (co)complete and p strictly preserves (co)limits.*

▶ **Example 2.5** ($\mathbf{CLat}_\wedge$-fibration).

- ($\mathbf{Pre} \to \mathbf{Set}$, $\mathbf{Pred} \to \mathbf{Set}$) These forgetful functors are $\mathbf{CLat}_\wedge$-fibrations. Here $\mathbf{Pre}$ is the category of preordered sets $(X, \leq_X)$ and order-preserving functions between them. $\mathbf{Pred}$ is that of predicates: objects are $P \subseteq X$, and morphisms $f : (P \subseteq X) \to (Q \subseteq Y)$ are functions $f : X \to Y$ satisfying $f(P) \subseteq Q$.
- ($\mathbf{ERel} \to \mathbf{Set}$) The functor $\mathbf{ERel} \to \mathbf{Set}$ defined by the change-of-base [19], as shown below, is a $\mathbf{CLat}_\wedge$-fibration. Concretely, $\mathbf{ERel}$ is the category of sets with binary relations $(X, R \subseteq X \times X)$ as objects, and relation-preserving maps as morphisms.

$$\begin{tikzcd}
\mathbf{ERel} \arrow[r] \arrow[d] & \mathbf{Pred} \arrow[d] \\
\mathbf{Set} \arrow[r, "(-)^2"] & \mathbf{Set}
\end{tikzcd}$$

- (Domain fibration $d^{\Omega} : \mathbf{Set}/\Omega \to \mathbf{Set}$) For each complete lattice Ω, we introduce a $\mathbf{CLat}_{\wedge}$-fibration $d^{\Omega} : \mathbf{Set}/\Omega \to \mathbf{Set}$ defined as follows.

  $$\begin{tikzcd} X \arrow[rr, "h"] \arrow[dr, "f"'] \arrow[rr, phantom, "\leq_X"', shift right=1.5ex] & & Y \arrow[dl, "g"] \\ & \Omega & \end{tikzcd}$$

  Here, $\mathbf{Set}/\Omega$ is a lax slice category defined as follows: objects of $\mathbf{Set}/\Omega$ are pairs $(X, f : X \to \Omega)$ of a set and a function (an "Ω-valued predicate on X"); we shall often write simply $f : X \to \Omega$ for the pair (X, f). Its morphisms from $f : X \to \Omega$ to $g : Y \to \Omega$ are functions $h : X \to Y$ such that $f \leq_X g \circ h$, as shown above.
  Then d_{Ω} is the evident forgetful functor, extracting the upper part of the above triangle. The order $\leq_X$ used there is the pointwise order between functions of the type $X \to \Omega$; the same order $\leq_X$ defines the order in each fiber $(\mathbf{Set}/\Omega)_X = \mathbf{Set}(X, \Omega)$. Following [4, Def. 4.1], we call d^{Ω} a *domain fibration* (from the lax slice category).

2.2 Chain Construction of Initial Algebras

▶ **Definition 2.6** (chain-cocomplete category)**.** A category $\mathbb{C}$ is *chain-cocomplete* if $\mathbb{C}$ has a colimit of every chain. We write 0 for a colimit of the empty chain (i.e. an initial object).

Noteworthy is that chain-cocompleteness is equivalent to existence of an initial object and filtered colimits, see [2, Cor. 1.7] for further details.

▶ **Definition 2.7** (initial chain [1], [3, Def. 3.2])**.** Let $\mathbb{C}$ be a chain-cocomplete category, and $F : \mathbb{C} \to \mathbb{C}$ be an endofunctor. The *initial chain* of F is the following diagram:

$$0 \xrightarrow{\alpha_{0,1}} F0 \xrightarrow{\alpha_{1,2}} \cdots \longrightarrow F^{\lambda}0 \xrightarrow{\alpha_{\lambda,\lambda+1}} \cdots . \tag{1}$$

This consists of the following.

- (Objects) It has objects $F^{i}0$ for each $i \in \mathrm{Ord}$ (where Ord is the category of ordinals), defined by $F^{0}0 = 0$, $F^{i+1}0 = F(F^{i}0)$, and for a limit ordinal i, $F^{i}0 = \mathrm{colim}_{j<i} F^{j}0$.
- (Morphisms) It has morphisms $\alpha_{i,j} : F^{i}0 \to F^{j}0$ for all ordinals i, j such that $i \leq j$, defined inductively on i. (Base case) $\alpha_{0,j} : 0 \to F^{j}0$ is the unique morphism. (Step case) $\alpha_{i+1,j+1}$ is $F\alpha_{i,j}$; for a limit ordinal j, $\alpha_{i+1,j}$ is from the colimiting cocone for $F^{j}0$. (Limit case) When i is a limit ordinal, $\alpha_{i,j}$ is induced by universality of $F^{i}0 = \mathrm{colim}_{k<i} F^{k}0$.

If $\alpha_{\lambda,\lambda+1}$ is an isomorphism, then we say that the initial chain of F *converges* in λ steps.

▶ **Proposition 2.8** (from [1], [3, Thm. 3.5])**.** *In the setting of Def. 2.7, assume that the initial chain converges in λ steps. Then $\alpha_{\lambda,\lambda+1}^{-1} : F^{\lambda+1}0 \xrightarrow{\cong} F^{\lambda}0$ is an initial F-algebra.*

The dual of the initial chain in Def. 2.7 is called *the final chain*. This also satisfies the dual of Prop. 2.8 (yielding final coalgebras), see [3, Def. 3.20 and Thm. 3.21].

The converse of Prop. 2.8 holds if we restrict to $\mathbf{Set}$.

▶ **Proposition 2.9** (from [30], [3, Cor. 3.16])**.** *A set functor has an initial algebra if and only if the initial chain converges.*

We often write μF for the carrier of an initial algebra of F.

The next basic lemma is important for us. Its dual (for coalgebras) is observed e.g. in [14].

▶ **Lemma 2.10.** *Assume that $p : \mathbb{E} \to \mathbb{B}$ is a fibration, that both $\mathbb{E}$ and $\mathbb{B}$ are chain-cocomplete, and that p strictly preserves chain colimits. Let $\dot F$ be a lifting of $F \colon \mathbb{B} \to \mathbb{B}$ along p.*

Consider the following initial chains.

$$\begin{tikzcd}
\mathbb{E} \arrow[d, "p"'] & 0 \arrow[r, "\dot\alpha_{0,1}"] & \dot F 0 \arrow[r, "\dot\alpha_{1,2}"] & \cdots \arrow[r] & \dot F^{\lambda} 0 \arrow[r, "\dot\alpha_{\lambda,\lambda+1}"] & \cdots \\
\mathbb{B} & 0 \arrow[r, "\alpha_{0,1}"] & F 0 \arrow[r, "\alpha_{1,2}"] & \cdots \arrow[r] & F^{\lambda} 0 \arrow[r, "\alpha_{\lambda,\lambda+1}"] & \cdots
\end{tikzcd}$$

1. *We have $\alpha_{i,j} = p\dot\alpha_{i,j}$ for all ordinals i, j with $i < j$.*
2. *Moreover, if the upper initial chain for $\dot F$ converges and yield an initial $\dot F$-algebra $\dot\alpha \colon \dot F(\mu\dot F) \to \mu\dot F$, then $p\dot\alpha \colon Fp(\mu\dot F) \to p(\mu\dot F)$ is an initial F-algebra.*

3 Initial Algebra-Final Coalgebra Coincidence over Initial Algebras

In this section, we formulate our target coincidence called the *IF/I coincidence*. It is a fibrational IA-FC coincidence over an initial algebra.

▶ **Definition 3.1** (IF/I coincidence)**.** Let $p : \mathbb{E} \to \mathbb{B}$ be a fibration, and $\dot F$ be a lifting of F. We say that the tuple $\bigl(p \colon \mathbb{E} \to \mathbb{B},\ F \colon \mathbb{B} \to \mathbb{B},\ \dot F \colon \mathbb{E} \to \mathbb{E}\bigr)$ satisfies the *IA-FC coincidence over an initial algebra (IF/I coincidence, for short)* if the following is satisfied.

1. There is an initial F-algebra $\beta \colon F(\mu F) \xrightarrow{\cong} \mu F$.
2. There is an initial $\dot F$-algebra $\dot\beta \colon \dot F(\mu\dot F) \xrightarrow{\cong} \mu\dot F$ above β.
3. Moreover, $\dot\beta^{-1}$ is final over β^{-1} in the following sense: for each $\dot F$-coalgebra γ above β^{-1} (shown below diagram on the left), there exists a unique vertical coalgebra morphism f from γ to $\dot\beta^{-1}$ (below diagram on the right, where vertical means $pf = \mathrm{id}_{\mu F}$).

$$\begin{tikzcd}
\mathbb{E} \arrow[loop left, "\dot F"] \arrow[dd, "p"'] & & & & \dot F(\mu\dot F) & \mu\dot F \arrow[l, "\dot\beta^{-1}"', "\cong"] \\
& \dot F P & P \arrow[l, "\gamma"'] & \Longrightarrow & \dot F P \arrow[u, dotted, "\dot F f"] & P \arrow[l, "\gamma"'] \arrow[u, dotted, "f"'] \\
\mathbb{B} \arrow[loop left, "F"] & F(\mu F) & \mu F; \arrow[l, "\beta^{-1}"', "\cong"] & & F(\mu F) & \mu F; \arrow[l, "\beta^{-1}"', "\cong"]
\end{tikzcd}$$

IF/I Coincidence in Fibrations of (Co)algebras. The IF/I coincidence in Def. 3.1 is nicely organized in terms of *fibrations of (co)algebras*: the last two conditions in Def. 3.1 can be stated succinctly in advanced fibrational terms.

Given a functor $F : \mathbb{B} \to \mathbb{B}$, $\mathbf{Alg}(F)$ is the category of F*-algebras*, where an object is a pair $(X \in \mathbb{B}, a \colon FX \to X)$ and a morphism from (X, a) to (Y, b) is $f \colon X \to Y$ such that $b \circ Ff = f \circ a$. Dually, $\mathbf{Coalg}(F)$ is the category of F*-coalgebras*, where an object is $(X \in \mathbb{B}, c \colon X \to FX)$ and a morphism from (X, c) to (Y, d) is f such that $d \circ f = Ff \circ c$.

Then a fibration p and a fibred lifting $\dot F$ yield fibrations of (co)algebras.

▶ **Proposition 3.2** (from [14, Prop. 4.1])**.** *A lifting $\dot F$ of F along a fibration p induces functors* $\mathbf{Alg}(p) : \mathbf{Alg}(\dot F) \to \mathbf{Alg}(F)$ *and* $\mathbf{Coalg}(p) : \mathbf{Coalg}(\dot F) \to \mathbf{Coalg}(F)$*, given by*

$$\begin{aligned}
\mathbf{Alg}(p) &: (\dot F P \xrightarrow{q} P) \longmapsto (FpP = p\dot F P \xrightarrow{pq} pP), \\
\mathbf{Coalg}(p) &: (P \xrightarrow{r} \dot F P) \longmapsto (pP \xrightarrow{pr} p\dot F P = FpP).
\end{aligned}$$

The functor $\mathbf{Alg}(p)$ *is a fibration. If additionally $\dot F$ is a fibred lifting, then* $\mathbf{Coalg}(p)$ *is a fibration, too. For an opfibration p, we have a result dual to the above:* $\mathbf{Coalg}(p)$ *is an opfibration; so is* $\mathbf{Alg}(p)$ *if $\dot F$ is an opfibred lifting (preserving cocartesian morphisms).*

The functor $\mathbf{Coalg}(p)\colon \mathbf{Coalg}(\dot{F}) \to \mathbf{Coalg}(F)$ in Prop. 3.2 plays an important role in the following development. It is thought of as a functor where

- (following the coalgebraic tradition) state-based transition systems $c\colon X \to FX$ and behavior-preserving morphisms between them populate the base category $\mathbf{Coalg}(F)$, and
- *invariants* – i.e. predicates $P \in \mathbb{E}_X$ over X that are *preserved* by transitions – populate the total category $\mathbf{Coalg}(\dot{F})$. The arrows in $\mathbf{Coalg}(\dot{F})$ are logical implication of invariants.

The following reformulation is proved in [21, Appendix C.2], together with technical remarks.

▶ **Proposition 3.3.** *The following is equivalent to Cond. 2 and 3 in Def. 3.1, respectively.*
2'. *There is an initial object $\dot{\beta}$ in the fiber $\mathbf{Alg}(\dot{F})_{\beta}$.*
3'. *$\dot{\beta}^{-1}$ is a final object in the fiber $\mathbf{Coalg}(\dot{F})_{\beta^{-1}}$.*

IF/I Coincidence in a CLat$_\wedge$-fibration. Here we shall rewrite conditions in Def. 3.1 for $\mathbf{CLat}_\wedge$-fibrations. But first, we need the following investigation of these conditions.

An initial $\dot{F}$-algebra lying above an initial F-algebra is a norm (Cond. 1–2; cf. Lem. 2.10). What is special is the finality of the initial $\dot{F}$-algebra (Cond. 3). The intuition of the latter is the following:

an lfp and a gfp coincide, in the fiber over the base initial algebra.

Intuitively, P with $(\gamma : P \to \dot{F}P) \in \mathbf{Coalg}(\dot{F})_{\beta^{-1}}$ is an *invariant* – it is a predicate that is preserved by the transition β^{-1}. Indeed, the morphism γ is equivalent to a morphism

$$\gamma^{\dagger}\colon P \longrightarrow (\beta^{-1})^{*}\dot{F}P \quad \text{over } \mathrm{id}_{\mu F}\text{, that is,}$$
$$P \le (\beta^{-1})^{*}\dot{F}P \quad \text{if the fibration } p\colon \mathbb{E} \to \mathbb{B} \text{ is posetal,}$$

by pulling back along β^{-1}. The latter inequality signifies that P is an invariant. This equivalence is formulated as follows.

▶ **Lemma 3.4.** *Let $p : \mathbb{E} \to \mathbb{B}$ be a fibration and $\dot{F}$ be a lifting of F along p. For any isomorphism $\alpha : X \xrightarrow{\cong} FX$ in $\mathbb{B}$, $\mathbf{Alg}(\dot{F})_{\alpha^{-1}} \cong \mathbf{Alg}(\alpha^{*}\dot{F})$ and $\mathbf{Coalg}(\dot{F})_{\alpha} \cong \mathbf{Coalg}(\alpha^{*}\dot{F})$.*

Therefore, Cond. 3 requires that $\dot{\beta}^{-1}$ gives a greatest invariant. In view of the Knaster–Tarski theorem (that a greatest post-fixed point is a greatest *fixed* point), this means that $\dot{\beta}^{-1}$ is a gfp if p is a $\mathbf{CLat}_\wedge$-fibration. Symmetrically, Cond. 2 (rephrased as Cond. 2') requires that $\dot{\beta}$ is an lfp. Therefore, the IF/I coincidence yields a coincidence between an lfp and a gfp. This plays an important role in the next section.

▶ **Proposition 3.5.** *If p is a $\mathbf{CLat}_\wedge$-fibration then Cond. 2 and 3 in Def. 3.1 are equivalent to the following condition: there is a unique fixed-point $\mu\dot{F}$ of $(\beta^{-1})^{*}\dot{F} : \mathbb{E}_{\mu F} \to \mathbb{E}_{\mu F}$.*

IF/I Coincidence over Base IA-FC Coincidence. The IF/I coincidence (Def. 3.1) allows a simpler formulation in the special case where the IA-FC coincidence is *already there in the base category.* In this case, $\dot{\beta}^{-1}$ is final not only in a suitable fiber (Cond. 3 of Def. 3.1; cf. Prop. 3.3), but also *globally* in the total category $\mathbb{E}$. See [21, Appendix C.4] for details.

This special setting (the base IA-FC coincidence) is known to hold in domain-theoretic settings [13,27]. We use this setting (specifically the IA-FC coincidence in a Kleisli category [15]) in one of our applications (§7).

▶ **Theorem 3.6** (IF/I coincidence over the base coincidence)**.** *Let $p : \mathbb{E} \to \mathbb{B}$ be a bifibration and $(p, F, \dot{F})$ be a tuple satisfying the IF/I coincidence. If there exists initial F-algebra $\beta : F(\mu F) \cong \mu F$ (in $\mathbb{B}$) such that β^{-1} is a final F-coalgebra, then there exists an initial $\dot{F}$-algebra $\dot{\beta} : \dot{F}(\mu\dot{F}) \cong \mu\dot{F}$ (in $\mathbb{E}$) above β such that $\dot{\beta}^{-1}$ is a final $\dot{F}$-coalgebra.*

4 IF/I Coincidence from Stable Chain Colimits

We now present our main observation, namely that the IF/I coincidence is a general phenomenon that relies only on a few mild assumptions. These assumptions include 1) that $\dot{F}$ is fibred (Def. 2.2) and 2) *stability* of chain colimits (Def. 4.1).

Here in §4, we restrict the underlying fibration $p\colon \mathbb{E} \to \mathbb{B}$ to a $\mathbf{CLat}_\wedge$-fibration over $\mathbf{Set}$ (Def. 2.3). This restriction simplifies proofs and technical developments. Nevertheless, we have a general coincidence theorem for not necessarily posetal fibrations; it is found in Appendix A. The general proof hinges on stable chain colimits, too.

The following is a key assumption. It is a fibrational adaptation of *pullback-stable colimit*, a notion studied in (higher) topos theory and categorical logic [9, 16, 22].

▶ **Definition 4.1** (stable chain colimits)**.** We say that a fibration $p : \mathbb{E} \to \mathbb{B}$ has *stable chain colimits* if the following condition holds: for each $\lambda \in \mathrm{Ord}$ and each diagram $D : \mathrm{Ord}_{<\lambda} \to \mathbb{B}$,

1. $\mathbb{B}$ has a colimit of D. The i-th cocone component is denoted by $\kappa_i\colon Di \to \operatorname{colim} D$.
2. Moreover, for each object $P \in \mathbb{E}_{\operatorname{colim} D}$ above $\operatorname{colim} D$, we have $P \cong \operatorname{colim} \kappa_i^* P$, with the cartesian liftings $(\kappa_i^* P \to P)_{i \in \mathrm{Ord}_{<\lambda}}$ forming a colimiting cocone.

$$\begin{tikzcd}
\mathbb{E} \arrow[d, "p"] & \kappa_0^* P \arrow[r] & \kappa_1^* P \arrow[r] & \cdots \arrow[r] & P \ (\cong \operatorname{colim} \kappa_i^* P) \\
\mathbb{B} & D0 \arrow[r] & D1 \arrow[r] & \cdots \arrow[r] & \operatorname{colim} D
\end{tikzcd}$$

The first condition is equivalent to chain-cocompleteness. The situation of the second condition is illustrated as the above diagram. Stability requires that the upper cocone is colimiting. In the diagram, we note that morphisms $\kappa_i^* P \to \kappa_j^* P$ above $D(i \to j)$ are well-defined (where $i \leq j \leq \lambda$); they are induced by universality of the cartesian liftings $\kappa_j^* P \to P$.

Letting $\lambda = 0$ in Def. 4.1 yields the following property.

▶ **Lemma 4.2.** *Let $p\colon \mathbb{E} \to \mathbb{B}$ be a fibration with stable chain colimits. Then, all objects in $\mathbb{E}_0$ are initial in $\mathbb{E}$.*

▶ **Example 4.3.** The fibrations in Example 2.5 – $\mathbf{Pre} \to \mathbf{Set}$, $\mathbf{Pred} \to \mathbf{Set}$, $\mathbf{ERel} \to \mathbf{Set}$, and the domain fibration d^Ω for any complete lattice Ω – all have stable chain colimits. Non-examples are deferred to [21, Appendix B].

▶ **Theorem 4.4** (Main result)**.** *Let $p : \mathbb{E} \to \mathbf{Set}$ be a $\mathbf{CLat}_\wedge$-fibration and $\dot{F}$ be a lifting of F along p. Assume further the following conditions:*

1. *there exists an initial F-algebra;*
2. *$\dot{F}$ is a fibred lifting of F;*
3. *p has stable chain colimits.*

Then $(p, F, \dot{F})$ satisfies the IF/I coincidence (Def. 3.1).

We prove the theorem in the rest of the section. Due to Prop. 3.5, it suffices to show that $(\beta^{-1})^* \dot{F}\colon \mathbb{E}_{\mu F} \to \mathbb{E}_{\mu F}$ has a unique fixed point, where β is an initial F-algebra. Cond. 1 in Thm. 4.4 yields that the initial chain of F converges and gives an initial F-algebra (Prop. 2.8 and 2.9).

We analyze the initial chains of F and $\dot{F}$, which is shown on the below.

$$\begin{tikzcd}
\mathbb{E} \arrow[d, "p"'] & 0 \arrow[r, "\dot{\alpha}_{0,1}"] & \dot{F}0 \arrow[r, "\dot{\alpha}_{1,2}"] & \cdots \arrow[r] & \dot{F}^\lambda 0 \arrow[r, "\dot{\alpha}_{\lambda,\lambda+1}"] & \dot{F}^{\lambda+1} 0 \arrow[r] & \cdots \\
\mathbf{Set} & 0 \arrow[r, "\alpha_{0,1}"] & F0 \arrow[r, "\alpha_{1,2}"] & \cdots \arrow[r] & F^\lambda 0 \arrow[r, "\alpha_{\lambda,\lambda+1}"] & \dot{F}^{\lambda+1} 0 \arrow[r] & \cdots
\end{tikzcd}$$

Prop. 2.4 and Lem. 2.10 ensure that each chain morphism $\dot{\alpha}_{i,i+1}$ is above $\alpha_{i,i+1}$. Then, assuming that the initial chain of F converges in λ steps, the functor $(\beta^{-1})^*\dot{F}$ of our interest is equal to $\alpha^*_{\lambda,\lambda+1}\dot{F}$.

Fig. 1 is the key diagram about a unique fixed-point of $\alpha^*_{\lambda,\lambda+1}\dot{F}$. For simplicity, we write α for $\alpha_{\lambda,\lambda+1}$. We find the initial chain of $\dot{F}$ as its middle row; the initial chain of $\alpha^*\dot{F}$ as the bottom half of the last column; and the final chain of $\alpha^*\dot{F}$ as the top half. The other objects in the diagram are obtained by applying substitution to the last column.

$$\begin{tikzcd}
\alpha^*_{0,\lambda}\top \arrow[r] \arrow[d, equal] & \alpha^*_{1,\lambda}\top \arrow[r] & \alpha^*_{2,\lambda}\top \arrow[r] & \cdots \arrow[r] & \top & \\
\alpha^*_{0,\lambda}\alpha^*\dot{F}\top \arrow[r] \arrow[d, equal] & \alpha^*_{1,\lambda}\alpha^*\dot{F}\top \arrow[r] \arrow[u] \arrow[d, equal] & \alpha^*_{2,\lambda}\alpha^*\dot{F}\top \arrow[r] \arrow[u] & \cdots \arrow[r] & \alpha^*\dot{F}\top \arrow[u] & \\
\alpha^*_{0,\lambda}(\alpha^*\dot{F})^2\top \arrow[r] \arrow[d, equal] & \alpha^*_{1,\lambda}(\alpha^*\dot{F})^2\top \arrow[r] \arrow[d, equal] & \alpha^*_{2,\lambda}(\alpha^*\dot{F})^2\top \arrow[r] \arrow[u] \arrow[d, equal] & \cdots \arrow[r] & (\alpha^*\dot{F})^2\top \arrow[u] & \\
\vdots & \vdots & \vdots & & \vdots \arrow[u] & \\
0 \arrow[r, "\dot{\alpha}_{0,1}"] & \dot{F}0 \arrow[r, "\dot{\alpha}_{1,2}"] & \dot{F}^20 \arrow[r] & \cdots \arrow[r] & \dot{F}^\lambda 0 & \text{(the initial } \dot{F}\text{-chain)} \\
\vdots \arrow[d, equal] & \vdots \arrow[d, equal] & \vdots \arrow[d, equal] & & \vdots & \\
\alpha^*_{0,\lambda}(\alpha^*\dot{F})^2\bot \arrow[r] \arrow[d, equal] & \alpha^*_{1,\lambda}(\alpha^*\dot{F})^2\bot \arrow[r] \arrow[d, equal] & \alpha^*_{2,\lambda}(\alpha^*\dot{F})^2\bot \arrow[r] & \cdots \arrow[r] & (\alpha^*\dot{F})^2\bot \arrow[u] & \\
\alpha^*_{0,\lambda}\alpha^*\dot{F}\bot \arrow[r] \arrow[d, equal] & \alpha^*_{1,\lambda}\alpha^*\dot{F}\bot \arrow[r] & \alpha^*_{2,\lambda}\alpha^*\dot{F}\bot \arrow[r] \arrow[u] & \cdots \arrow[r] & \alpha^*\dot{F}\bot \arrow[u] & \\
\alpha^*_{0,\lambda}\bot \arrow[r] & \alpha^*_{1,\lambda}\bot \arrow[r] \arrow[u] & \alpha^*_{2,\lambda}\bot \arrow[r] \arrow[u] & \cdots \arrow[r] & \bot \arrow[u] & \\
 & & & & & \\
0 \arrow[r, "\alpha_{0,1}"] & F0 \arrow[r, "\alpha_{1,2}"] & F^20 \arrow[r] & \cdots \arrow[r] & F^\lambda 0 & \text{(the initial } F\text{-chain)}
\end{tikzcd}$$

Figure 1 IA-FC coincidence for $\mathbf{CLat}_\wedge$-fibrations, in Prop. 4.5. Here we write α for $\alpha_{\lambda,\lambda+1}$; the choice of λ is arbitrary (the initial $\dot{F}$-chain may not have stabilized).

The next result is the key technical observation. It says 1) the upper rows become closer to the initial $\dot{F}$-chain as we go below; and 2) symmetrically, the lower rows become closer to the same as we go up. Its proof is by transfinite induction; the stability assumption is crucially used in its limit case.

▶ **Proposition 4.5.** *Consider the setting of Thm. 4.4. Let λ be an arbitrary ordinal. We write $\alpha, \dot{\alpha}$ for $\alpha_{\lambda,\lambda+1}, \dot{\alpha}_{\lambda,\lambda+1}$ and $\top, \bot$ for the maximum and minimum of the complete lattice $\mathbb{E}_{F^\lambda 0}$. For each ordinal i, the objects $(\alpha^*\dot{F})^i\bot$ and $(\alpha^*\dot{F})^i\top$ above $F^\lambda 0$ are defined by the initial chain and the final chain of $\alpha^*\dot{F}$ (the last column of Fig. 1).*

Then we have $\alpha^*_{i,\lambda}(\alpha^*\dot{F})^i\bot = \dot{F}^i 0 = \alpha^*_{i,\lambda}(\alpha^*\dot{F})^i\top$ for each i with $i \leq \lambda$.

Proof sketch; a full proof is in [21, Appendix C.6]. The proof is by transfinite induction on i. The base case is clear because $\mathbb{E}_0$ includes only one object by Lemma 4.2 and the posetal assumption on p.

In the step case, fibredness of the lifting $\dot{F}$ lifts the equality for i, which is $\alpha^*_{i,\lambda}(\alpha^*\dot{F})^i\bot = \dot{F}^i 0 = \alpha^*_{i,\lambda}(\alpha^*\dot{F})^i\top$ (the induction hypothesis), to the desired equality for $i+1$.

The limit case is less obvious than the other cases. We rewrite the target objects (e.g. $\alpha^*_{i,\lambda}(\alpha^*\dot{F})^i\bot$) to chain colimits (e.g. $\mathrm{colim}_{j<i}\,\alpha^*_{j,\lambda}(\alpha^*\dot{F})^i\bot$) by stability of chain colimits, and use the fact that colimits of diagonal elements (e.g. $\mathrm{colim}_{j<i}\,\alpha^*_{j,\lambda}(\alpha^*\dot{F})^j\bot$) are equal to $\dot{F}^i 0$ by the induction hypothesis. See [21, Appendix C.6] for a full proof. ◀

Letting $i = \lambda$ in Prop. 4.5 yields that $(\alpha^*\dot{F})^\lambda\bot = \dot{F}^\lambda 0 = (\alpha^*\dot{F})^\lambda\top$. Therefore, both the initial and final chains of $\alpha^*\dot{F}$ (the last column in Fig. 1) converge in λ steps. We conclude that $\dot{F}^\lambda 0$ is both the lfp and gfp for $\alpha^*\dot{F}\colon \mathbb{E}_{F^\lambda 0} \to \mathbb{E}_{F^\lambda 0}$, hence is its unique fixed point.

Here are some consequences of the proposition. In the next result, note that the number of converging steps of F and that of $\dot{F}$ are not the same in general. See [21, Appendix B] for an example.

▶ **Corollary 4.6.** *Let $p : \mathbb{E} \to \mathbf{Set}$ be a $\mathbf{CLat}_\wedge$-fibration and $\dot{F}$ be a fibred lifting of F along p. Assume p has stable chain colimits. Then, the initial chain of F converges in λ steps if and only if that of $\dot{F}$ converges in λ steps.*

▶ **Corollary 4.7.** *In the setting of Cor. 4.6, if F has an initial algebra α, then any isomorphism $\dot{F}P \to P$ above α is an initial algebra of $\dot{F}$.*

We are finally in a position to prove our main theorem.

Proof of Thm. 4.4. Using Prop. 2.9 and Cor. 4.6, Cond. 1 ensures the existence of an ordinal λ such that both the initial chains of F and $\dot{F}$ converges in the steps. Then $\alpha^{-1}_{\lambda,\lambda+1}$ is an initial F-algebra by Prop. 2.8 and $\dot{F}^\lambda 0$ is a unique fixed-point of $\alpha^*_{\lambda,\lambda+1}\dot{F}$ by Prop. 4.5. Prop. 3.5 concludes the proof. ◀

5 Coincidence for Ω-Valued Predicates, Pure and Effectful

We instantiate the above categorical results to an important family of examples, namely Ω-*valued predicates* (Example 2.5). In this setting, a functor lifting $\dot{F}$ (§3) has a concrete presentation as an F-algebra, an observation that helps identification of many examples.

Besides the "pure" setting modeled by the fibration $\mathbf{Set}/\Omega \to \mathbf{Set}$, we also consider the "effectful" setting $\mathbf{Kl}(\dot{\mathcal{T}}) \to \mathbf{Kl}(\mathcal{T})$, where effects are modeled by a monad $\mathcal{T}$ [23] with its lifting $\dot{\mathcal{T}}$ along d^Ω, and the base category is the Kleisli category for $\mathcal{T}$. The categorical construction of the fibration $\mathbf{Kl}(\dot{\mathcal{T}}) \to \mathbf{Kl}(\mathcal{T})$ is described later in §5.2; the construction builds upon the recent results in [4].

The theoretical development here in §5 specializes that in §3–4, but it is still in abstract categorical terms. The theory in §5 paves the way to the concrete applications in §6–7.

5.1 Coincidence for Ω-Valued Predicates, the Pure Setting

We first focus on the domain fibration $d^\Omega : \mathbf{Set}/\Omega \to \mathbf{Set}$ (Example 2.5), where 1) a complete lattice Ω is regarded as a truth value domain, and 2) the fibration is regarded as that of Ω-valued predicates. If Ω is the two-element lattice $\mathbf{2} = \{\bot, \top\}$, then $d^{\mathbf{2}} : \mathbf{Set}/\mathbf{2} \to \mathbf{Set}$ is isomorphic to $\mathbf{Pred} \to \mathbf{Set}$.

Towards the IF/I coincidence for the fibration d^Ω, we first need to describe a fibred lifting $\dot{F}$ of F. It is induced by an F-algebra over Ω that is equipped with a suitable order structure.

▶ **Definition 5.1** (monotone algebra [4]). Let $F : \mathbf{Set} \to \mathbf{Set}$ be a functor and Ω be a complete lattice. We call $\sigma : F\Omega \to \Omega$ a *monotone F-algebra* over Ω if $i \leq_X i' \Rightarrow \sigma \circ Fi \leq_{FX} \sigma \circ Fi'$ holds for all $X \in \mathbf{Set}$ and all $i, i' \in \mathbf{Set}(X, \Omega)$.

▶ **Lemma 5.2** (from [4, 7]). *Let $F : \mathbf{Set} \to \mathbf{Set}$ be a functor, and Ω be a complete lattice. There is a bijective correspondence between monotone F-algebras $\sigma : F\Omega \to \Omega$ and fibred liftings $\dot{F}$ of F along d^Ω. Specifically, σ gives rise to the lifting $\dot{F}$ given by $\dot{F}(X \xrightarrow{x} \Omega) = (FX \xrightarrow{Fx} F\Omega \xrightarrow{\sigma} \Omega)$; conversely, $\dot{F}$ gives rise to $(F\Omega \xrightarrow{\sigma} \Omega) = \dot{F}(\Omega \xrightarrow{\mathrm{id}_\Omega} \Omega)$.*

Application of §4 to a domain fibration is then easy.

▶ **Theorem 5.3** (coincidence for Ω-valued predicates, pure). *In the setting of Def. 5.1, let $\sigma : F\Omega \to \Omega$ be a monotone F-algebra. By Lem. 5.2, σ gives rise to a fibred lifting $\dot{F}$ of F along d^Ω. If there exists an initial F-algebra then $(d^\Omega, F, \dot{F})$ satisfies the IF/I coincidence.*

5.2 Coincidence for Ω-Valued Predicates, Effectful

In order to accommodate some concrete examples (those in §7 to be specific), we extend the above material to the setting with monadic effects.

We aim at the situation in (2), where the domain fibration d^{Ω} is Kleisli-embedded in the fibration $d^{\Omega}_{\mathcal{T},\dot{\mathcal{T}}}\colon \mathbf{Kl}(\dot{\mathcal{T}}) \to \mathbf{Kl}(\mathcal{T})$ on the right. The latter is the desired fibration of effectful computations and Ω-valued predicates; moreover, we extend a functor F and its lifting $\dot{F}$ for the Kleisli fibration, too.

$$\begin{tikzcd}
\mathbf{Set}/\Omega \arrow[loop left, "\dot{F}"] \arrow[r, "\dot{L}"] \arrow[d, "d^{\Omega}"] & \mathbf{Kl}(\dot{\mathcal{T}}) \arrow[loop right, "\dot{F}_{\dot{\mathcal{T}}}"] \arrow[d, "d^{\Omega}_{\mathcal{T},\dot{\mathcal{T}}}"] \\
\mathbf{Set} \arrow[loop left, "F"] \arrow[r, "L"] & \mathbf{Kl}(\mathcal{T}) \arrow[loop right, "F_{\mathcal{T}}"]
\end{tikzcd} \tag{2}$$

The construction of the Kleisli fibration $d^{\Omega}_{\mathcal{T},\dot{\mathcal{T}}}$ is via a *cartesian lifting* of the monad $\mathcal{T}$. It is defined to be a monad $(\dot{\mathcal{T}}, \dot{\eta}, \dot{\mu})$ on $\mathbf{Set}/\Omega$ such that 1) $\dot{\mathcal{T}}$ (as a functor) is a fibred lifting of the functor $\mathcal{T}$, and 2) $\dot{\eta}, \dot{\mu}$ are componentwise cartesian morphisms above η, μ, respectively. Then $d^{\Omega}_{\mathcal{T},\dot{\mathcal{T}}} : \mathbf{Kl}(\dot{\mathcal{T}}) \to \mathbf{Kl}(\mathcal{T})$ is defined to be the evident extension of d^{Ω} to Kleisli categories, and is a fibration [4]. Cartesian liftings of $\mathcal{T}$ from $\mathbf{Set}$ to $\mathbf{Set}/\Omega$ bijectively correspond to Eilenberg-Moore (EM) $\mathcal{T}$-algebras, much like in Lem. 5.2.

▶ **Definition 5.4** (EM monotone algebra [4])**.** Let $\mathcal{T} : \mathbf{Set} \to \mathbf{Set}$ be a monad and Ω be a complete lattice. A monotone $\mathcal{T}$-algebra $\tau : \mathcal{T}\Omega \to \Omega$ (where $\mathcal{T}$ is identified with its underlying functor) is called an *Eilenberg-Moore (EM) monotone $\mathcal{T}$-algebra* if $\mathrm{id}_{\Omega} = \tau \circ \eta_{\Omega}$ and $\tau \circ \mathcal{T}\tau = \tau \circ \mu_{\Omega}$. Here η and μ are the unit and multiplication of the monad $\mathcal{T}$.

▶ **Lemma 5.5** (from [4, Thm. 4.4])**.** *Let $\mathcal{T} : \mathbf{Set} \to \mathbf{Set}$ be a monad and Ω be a complete lattice. There is a bijective correspondence between*

- *EM monotone $\mathcal{T}$-algebras τ, and*
- *Cartesian liftings $\dot{\mathcal{T}}$ of $\mathcal{T}$ that is itself a monad on $\mathbf{Set}/\Omega$.*

Specifically, τ gives rise to the lifting $\dot{\mathcal{T}}$ given by $\dot{\mathcal{T}}(X \xrightarrow{x} \Omega) = (\mathcal{T}X \xrightarrow{\mathcal{T}x} \mathcal{T}\Omega \xrightarrow{\tau} \Omega)$; conversely, $\dot{\mathcal{T}}$ gives rise to $(\mathcal{T}\Omega \xrightarrow{\tau} \Omega) = \dot{\mathcal{T}}(\Omega \xrightarrow{\mathrm{id}_{\Omega}} \Omega)$.

Let us now describe the fibration $d^{\Omega}_{\mathcal{T},\dot{\mathcal{T}}}$ between Kleisli categories – it is the one on the right in (2). Recall that the *Kleisli category* $\mathbf{Kl}(\mathcal{T})$ of a monad $\mathcal{T}$ on $\mathbb{C}$ has the same objects as $\mathbb{C}$, and its morphisms from C to D are $\mathbb{C}$-morphisms $C \to \mathcal{T}D$ (often denoted by $C \nrightarrow D$). In view of Lem. 5.5, the Kleisli category $\mathbf{Kl}(\dot{\mathcal{T}})$ is described as follows:

- its objects are pairs $(X, f : X \to \Omega)$ where the latter is an Ω-valued predicate;
- its morphisms from $(X, f : X \to \Omega)$ to $(Y, g : Y \to \Omega)$ are $h : X \to \mathcal{T}Y$ such that $f \leq_X \dot{\mathcal{T}}g \circ h$ as shown below, where τ is the EM monotone $\mathcal{T}$-algebra that corresponds to the lifting $\dot{\mathcal{T}}$ (Lem. 5.5).

$$\begin{tikzcd}
X \arrow[rr, "h"] \arrow[dr, "f"'] & \arrow[d, phantom, "\leq_X"] & \mathcal{T}Y \arrow[dl, "\dot{\mathcal{T}}g = \tau \circ \mathcal{T}g"] \\
 & \Omega &
\end{tikzcd}$$

▶ **Lemma 5.6** (the fibration $d^{\Omega}_{\mathcal{T},\dot{\mathcal{T}}}\colon \mathbf{Kl}(\dot{\mathcal{T}}) \to \mathbf{Kl}(\mathcal{T})$ [4, Cor. 3.5])**.** *Let $\mathcal{T} : \mathbf{Set} \to \mathbf{Set}$ be a monad, Ω be a complete lattice, and τ be an EM monotone $\mathcal{T}$-algebra. By Lem. 5.5, τ gives the fibred lifting $\dot{\mathcal{T}}$ of $\mathcal{T}$ such that $\dot{\mathcal{T}}$ is a monad.*

1. *The functor $d^{\Omega}_{\mathcal{T},\dot{\mathcal{T}}} : \mathbf{Kl}(\dot{\mathcal{T}}) \to \mathbf{Kl}(\mathcal{T})$, defined as follows, is a posetal fibration: $(X \to \Omega) \longmapsto X$ on objects, and $\bigl(f : (X \to \Omega) \nrightarrow (Y \to \Omega)\bigr) \longmapsto f : X \nrightarrow Y$ on morphisms.*
2. *For each X in $\mathbf{Set}$, we have the isomorphism $(\mathbf{Set}/\Omega)_X \cong \mathbf{Kl}(\dot{\mathcal{T}})_{LX}$ between fibers. Here $L : \mathbf{Set} \to \mathbf{Kl}(\mathcal{T})$ is the Kleisli left adjoint that carries each object X to X.*

Now that we have described the fibration $d^{\Omega}_{\mathcal{T},\dot{\mathcal{T}}} : \mathbf{Kl}(\dot{\mathcal{T}}) \to \mathbf{Kl}(\mathcal{T})$, let us extend the functors $F, \dot{F}$ to $F_{\mathcal{T}}, \dot{F}_{\dot{\mathcal{T}}}$ (cf. (2)). We can do so by specifying how F and $\mathcal{T}$ interact.

▶ **Definition 5.7** (distributive law [24])**.** Let $F : \mathbf{Set} \to \mathbf{Set}$ be a functor and $\mathcal{T} : \mathbf{Set} \to \mathbf{Set}$ be a monad with unit η and multiplication μ. A *distributive law* of F over $\mathcal{T}$ is a natural transformation $\lambda : F\mathcal{T} \Rightarrow \mathcal{T}F$ that makes the following diagrams commute.

$$\begin{tikzcd}
FX \arrow[r, "F\eta_X"] \arrow[dr, "\eta_{FX}"'] & F\mathcal{T}X \arrow[d, "\lambda_X"] \\
& \mathcal{T}FX
\end{tikzcd}
\qquad
\begin{tikzcd}
F\mathcal{T}^2X \arrow[r, "\lambda_{\mathcal{T}X}"] \arrow[d, "F\mu_X"] & \mathcal{T}F\mathcal{T}X \arrow[r, "\mathcal{T}\lambda_X"] & \mathcal{T}^2FX \arrow[d, "\mu_{FX}"] \\
F\mathcal{T}X \arrow[rr, "\lambda_X"] & & \mathcal{T}FX
\end{tikzcd}$$

▶ **Lemma 5.8** (from [24])**.** *Let $F : \mathbf{Set} \to \mathbf{Set}$ be a functor, $\mathcal{T} : \mathbf{Set} \to \mathbf{Set}$ be a monad and $L : \mathbf{Set} \to \mathbf{Kl}(\mathcal{T})$ be the left adjoint to the Kleisli category of $\mathcal{T}$. There is a bijective correspondence between distributive laws $\lambda : F\mathcal{T} \Rightarrow \mathcal{T}F$ and extensions $F_{\mathcal{T}} : \mathbf{Kl}(\mathcal{T}) \to \mathbf{Kl}(\mathcal{T})$ of F along L (that is, $F_{\mathcal{T}} \circ L = L \circ F$).*

The next lemma tells how to lift a distributive law λ of F over $\mathcal{T}$ to that of $\dot{F}$ over $\dot{\mathcal{T}}$. It follows from [4, Thm. 4.4].

▶ **Lemma 5.9.** *Let $F : \mathbf{Set} \to \mathbf{Set}$ be a functor, $\mathcal{T} : \mathbf{Set} \to \mathbf{Set}$ be a monad, and Ω be a complete lattice. Consider a fibred lifting $\dot{F}$ of F corresponding to a monotone F-algebra $\sigma : F\Omega \to \Omega$ and a Cartesian lifting $\dot{\mathcal{T}}$ of $\mathcal{T}$ corresponding to an EM monotone $\mathcal{T}$-algebra $\tau : \mathcal{T}\Omega \to \Omega$ (see Lem. 5.2 and 5.5). Assume further that a distributive law $\lambda : F\mathcal{T} \Rightarrow \mathcal{T}F$ is compatible with σ and τ, in the sense that $\sigma \circ F\tau \leq \tau \circ \mathcal{T}\sigma \circ \lambda_{\Omega}$. Then this λ induces a distributive law $\dot{\lambda} : \dot{F}\dot{\mathcal{T}} \Rightarrow \dot{\mathcal{T}}\dot{F}$ of $\dot{F}$ over $\dot{\mathcal{T}}$ above λ.*

Finally, we obtain the fibrations and functors shown in (2).

▶ **Definition 5.10.** Assume the setting of Thm. 5.3. Let $\mathcal{T} : \mathbf{Set} \to \mathbf{Set}$ be a monad; τ be an EM monotone $\mathcal{T}$-algebra on Ω; and λ be a distributive law satisfying $\sigma \circ F\tau \leq \tau \circ \mathcal{T}\sigma \circ \lambda_{\Omega}$. We define $(d^{\Omega}_{\mathcal{T},\dot{\mathcal{T}}}, F_{\mathcal{T}}, \dot{F}_{\dot{\mathcal{T}}})$ as follows.

- The EM monotone $\mathcal{T}$-algebra $\tau : \mathcal{T}\Omega \to \Omega$ gives rise to a Cartesian monad lifting $\dot{\mathcal{T}}$ of $\mathcal{T}$ along d^{Ω} (Lem. 5.5) and a fibration $d^{\Omega}_{\mathcal{T},\dot{\mathcal{T}}} : \mathbf{Kl}(\dot{\mathcal{T}}) \to \mathbf{Kl}(\mathcal{T})$ (Lem. 5.6).
- The distributive law $\lambda : F\mathcal{T} \Rightarrow \mathcal{T}F$ induces $F_{\mathcal{T}} : \mathbf{Kl}(\mathcal{T}) \to \mathbf{Kl}(\mathcal{T})$ such that $F_{\mathcal{T}}$ is an extension of F (in the sense of $F_{\mathcal{T}} \circ L = L \circ F$, Lem. 5.8).
- Because λ satisfies $\sigma \circ F\tau \leq \tau \circ \mathcal{T}\sigma \circ \lambda_{\Omega}$, Lem. 5.9 canonically induces a distributive law $\dot{\lambda} : \dot{F}\dot{\mathcal{T}} \Rightarrow \dot{\mathcal{T}}\dot{F}$.
- This distributive law $\dot{\lambda}$ induces an extension $\dot{F}_{\dot{\mathcal{T}}} : \mathbf{Kl}(\dot{\mathcal{T}}) \to \mathbf{Kl}(\dot{\mathcal{T}})$ of $\dot{F}$ (Lem. 5.8), which is also a lifting of $\dot{F}$ along $d^{\Omega}_{\mathcal{T},\dot{\mathcal{T}}}$.
- (Optional) If λ satisfies the equality $\sigma \circ F\tau = \tau \circ \mathcal{T}\sigma \circ \lambda_{\Omega}$ (instead of the inequality $\leq$ required in the above), then $\dot{F}_{\dot{\mathcal{T}}}$ is a fibred lifting of $F_{\mathcal{T}}$.

The above technical material (mainly from [4]) allows us to state this section's main result.

▶ **Theorem 5.11** (coincidence for Ω-valued predicates, effectful)**.** *In the setting of Def. 5.10, if there exists an initial F-algebra then $(d^{\Omega}_{\mathcal{T},\dot{\mathcal{T}}}, F_{\mathcal{T}}, \dot{F}_{\dot{\mathcal{T}}})$ satisfies the IF/I coincidence.*

The proof of Thm. 5.11 is *not* a straightforward application of the general results in §4 to the fibration $d^{\Omega}_{\mathcal{T},\dot{\mathcal{T}}}$. Notice, for example, that fibredness of the lifting $\dot{F}_{\dot{\mathcal{T}}}$ is not mandatory in Def. 5.10, while it is required in the general IF/I coincidence result (Thm. 4.4). Indeed, the lifting $\dot{F}_{\dot{\mathcal{T}}}$ is not fibred in our application in §7, so Thm. 4.4 does not apply to it.

Our proof of Thm. 5.11 instead goes via the “pure” fibration d^{Ω} (on the left in (2)): using the fact that the left adjoint L preserves initial chains, we essentially lift the IF/I coincidence from pure (d^{Ω}) to effectful ($d^{\Omega}_{\mathcal{T},\dot{\mathcal{T}}}$). We count this proof (in [21, Appendix C.10]) as one of our main contributions.

6 Application 1: Probabilistic Liveness by Submartingales

We use the IF/I coincidence results in §3–5 to derive a new proof method for probabilistic liveness – more concretely, we derive the method as an instance of Thm. 5.3. Liveness properties are usually witnessed by *ranking supermartingales*; see e.g. [10, 29]. Restricting to finite trees, we show that probabilistic liveness can also be witnessed by an invariant-like *sub*martingale (as opposed to *super*martingale) notion.

Here is the class of probabilistic systems that we analyze. It is restricted for the simplicity of presentation; accommodating more expressive formalisms is easy by changing a functor.

▶ **Definition 6.1** (finite probabilistic binary tree)**.** A *finite probabilistic binary tree* is a finite binary tree such that

- each internal node n is labeled with either ✓ or ?; and
- each edge is labeled with a real number $p \in [0,1]$, in such a way that two outgoing edge-labels sum to 1. See (3).

$$\begin{tikzcd}
 & n \arrow[dl, "p"'] \arrow[dr, "1-p"] & \\
n_1 & & n_2
\end{tikzcd} \tag{3}$$

We restrict to *finite* trees; here is one application scenario that justifies it. We think of those probabilistic trees as models of systems with internal coin toss. We assume that there is some timeout mechanism that forces the termination of those systems, that is, that termination of the target system is guaranteed by some external means. Such mechanism forcing finiteness is common in real-world systems.

The liveness property we are interested in is eventually reaching a state labeled with ✓. More precisely, we are interested in the probability of eventually seeing ✓. The following invariant-like witness notion gives a guaranteed lower bound for the probability in question. It is derived from the IF/I coincidence; unlike ranking supermartingales, it does not use natural numbers or ordinals.

▶ **Definition 6.2** (IF/I submartingale)**.** Let t be a finite probabilistic binary tree; the set of its nodes is denoted by N. We say $f : N \to [0,1]$ is an *IF/I submartingale* if it satisfies the following.

1. $f(n) = 0$ for each leaf node n.
2. For each internal node n labeled with ?, let its children and their edge labels be as shown in (3). Then we have

$$f(n) \le p \cdot f(n_1) + (1-p) \cdot f(n_2).$$

The direction of the inequality is indeed that of a *sub*martingale: the current value is a lower bound of the expected next value. Note that there is no condition for $f(n)$ if n is an internal node labeled with ✓. In this case, $f(n)$ can be set to 1 to improve the lower bound.

▶ **Theorem 6.3.** *In the setting of Def. 6.2, assume f is an IF/I submartingale. Then, identifying the tree t with the Markov chain with suitable probabilistic branching, the probability of eventually reaching a node labeled with ✓ from the root is at least $f(r_t)$ where r_t is the root node of t.*

The proof of Thm. 6.3 is in [21, Appendix C.11]. The main step is to apply the following to Thm. 5.3 in order to obtain a categorical data $(d^{[0,1]}, F^{\mathrm{ptr}}, \dot{F}^{\mathrm{ptr}})$ satisfying the IF/I coincidence:

- a complete lattice Ω is $[0,1]$ with the usual order between real numbers;
- a set functor F is $F^{\mathrm{ptr}} = \mathbf{1} + \{\checkmark, ?\} \times [0,1] \times (-)^2$; and
- a monotone F^{ptr}-algebra $\sigma : F^{\Sigma}[0,1] \to [0,1]$ is σ^{ptr} defined as follows:

$$\sigma^{\mathrm{ptr}}(x) = \begin{cases} 0 & \text{if } x = * \in \mathbf{1} \\ 1 & \text{if } x = (\checkmark, p, a, b) \\ pa + (1-p)b & \text{if } x = (?, p, a, b). \end{cases}$$

7 Application 2: Witnesses for Bottom-Up Tree Automata

We present an application of the IF/I coincidence in §3 to tree automata, using the results in §5 as an interface. In this paper we restrict to *bottom-up* tree automata, although a similar theory can be developed for top-down ones.

We restrict the ranked alphabet Σ used for trees to $\Sigma = \Sigma_0 \cup \Sigma_2$, where operations in Σ_0 are 0-ary and those in Σ_2 are binary. This restriction is for simplicity and not essential.

▶ **Definition 7.1** ((finite) Σ-trees). A Σ-*tree* t is a tuple (N, r_t, c_t) where N is a set of nodes, $r_t \in N$ is a root node and $c_t : N \to \Sigma_0 + \Sigma_2 \times N \times N$ is a function which determines labels and next nodes: if $c_t(n) = s \in \Sigma_0$ then n is a leaf node labeled with $s \in \Sigma_0$, and if $c_t(n) = (s, n_1, n_2)$ then n is an internal node labeled with $s \in \Sigma_2$ and the next nodes of n are n_1 and n_2. A *finite* Σ-*tree* is a Σ-tree which has only finitely many nodes.

▶ **Definition 7.2** (bottom-up tree automaton). A *bottom-up tree automaton* is a quadruple $\mathcal{A} = (\Sigma_0 \cup \Sigma_2, Q, \delta, q_{\mathsf{F}})$, where 1) $\Sigma_0 \cup \Sigma_2$ is a ranked alphabet; 2) Q is a set of states; 3) $\delta \colon \Sigma_0 + \Sigma_2 \times Q \times Q \to \mathcal{P}Q$ is a transition function (note the nondeterminism modeled by the powerset $\mathcal{P}Q$); and 4) $q_{\mathsf{F}} \in Q$ is an accepting state.

A *run* of $\mathcal{A}$ over a Σ-tree t is a function ρ from nodes n of t to states $\rho(n) \in Q$ such that 1) $\rho(n) \in \delta(s)$ for each leaf node n with $c_t(n) = s$, and 2) $\rho(n) \in \delta\big(s, \rho(n_1), \rho(n_2)\big)$ for each internal node n with $c_t(n) = (s, n_1, n_2)$.

A finite Σ-tree t is *accepted* by $\mathcal{A}$ if there is a run ρ of $\mathcal{A}$ over t such that $\rho(r_t) = q_{\mathsf{F}}$.

Note that allowing multiple accepting states does not change the theory because of the nondeterminism in transition functions.

Upside-Down Witness for Acceptance. For an acceptance of a single Σ-tree by a bottom-up tree automaton, the IF/I coincidence in §3 and §5.1 (the pure setting) yields the following (invariant-like, top-down) witness notion.

▶ **Definition 7.3.** Let $\mathcal{A} = (\Sigma_0 \cup \Sigma_2, Q, \delta, q_{\mathsf{F}})$ be a bottom-up tree automaton, and $t = (N, r_t, c_t)$ be a finite Σ-tree. We say $f : N \to \mathcal{P}Q$ is an *acceptance invariant* if

1. for each leaf node n with $c_t(n) = s$, we have $f(n) \subseteq \delta(s)$;
2. for each internal node n with $c_t(n) = (s, n_1, n_2)$, we have $f(n) \subseteq \bigcup_{q_1 \in f(n_1), q_2 \in f(n_2)} \delta\big(s, q_1, q_2\big)$;
3. for the root node r_t of t, we have $q_{\mathsf{F}} \in f(r_t)$.

An acceptance invariant assigns a *predicate* $f(n)$ to each node n, and the constraints on f runs in the top-down manner. The proof of Thm. 7.4 is in [21, Appendix C.12], where we identify suitable categorical constructs (a fibration and functors) and apply the results in §5.1.

▶ **Theorem 7.4** (acceptance witness for a finite tree)**.** *In the setting of Def. 7.3, if there exists an acceptance invariant $f : N \to \mathcal{P}Q$, then $\mathcal{A}$ accepts the finite Σ-tree t.*

Upside-Down Witness for Model Checking. We extend the above theory from acceptance (of a single tree) to *model checking* (whether every tree generated by a system is accepted). Besides its practical relevance, the model checking problem is categorically interesting. Specifically, for the results here, we use the extended categorical framework in §5.2 (IF/I coincidence in presence of *effects*) and Thm. 3.6 (coincidence lifting).

▶ **Definition 7.5** (generative tree automaton $\mathcal{C}$, its language $L_{\mathcal{C}}^{\mathrm{fin}}$, and model checking)**.** A *generative tree automaton* is $\mathcal{C} = (\Sigma_0 \cup \Sigma_2, X, c, x_0)$, where 1) $\Sigma_0 \cup \Sigma_2$ is a ranked alphabet; 2) X is a set of states; 3) $c\colon X \to \mathcal{P}(\Sigma_0 + \Sigma_2 \times X \times X)$ is a transition function (note the powerset operator $\mathcal{P}$); and 4) $x_0 \in X$ is an initial state.

Let $t = (N, r_t, c_t)$ be a (possibly infinite) Σ-tree. A *run* of $\mathcal{C}$ over t is a function $\rho\colon N \to X$, assigning a state to each node, such that 1) $\rho(r_t) = x_0$ for the root node r_t; 2) $s \in c(\rho(n))$ for each leaf node n with $c_t(n) = s$; and 3) $\bigl(s, \rho(n_1), \rho(n_2)\bigr) \in c(\rho(n))$ for each internal node n with $c_t(n) = (s, n_1, n_2)$.

We say that a Σ-tree t is *generated* by $\mathcal{C}$ if there is a run ρ of $\mathcal{C}$ over t. The set of all Σ-trees generated by $\mathcal{C}$ is denoted by $L_{\mathcal{C}}$; the set of all *finite* Σ-trees generated by $\mathcal{C}$ is $L_{\mathcal{C}}^{\mathrm{fin}}$.

The *model checking problem* takes a generative tree automaton $\mathcal{C}$ and a bottom-up tree automaton $\mathcal{A}$ (Def. 7.2) as input, and asks if every finite Σ-tree in $L_{\mathcal{C}}^{\mathrm{fin}}$ is accepted by $\mathcal{A}$.

Note that we restrict to finite trees here. One possible justification is an external mechanism that forces termination, much like in §6.

Our general theory of the IF/I coincidence derives the following (invariant-like, top-down) witness notion for model checking (where the specification is a bottom-up tree automaton).

▶ **Definition 7.6** (model checking invariant)**.** Let $\mathcal{A} = (\Sigma_0 \cup \Sigma_2, Q, \delta, q_{\mathsf{F}})$ be a bottom-up tree automaton, and let $\mathcal{C} = (\Sigma_0 \cup \Sigma_2, X, c, x_0)$ be a generative tree automaton. We say $f : X \to \mathcal{P}Q$ is a *model checking invariant* if it satisfies the following.

1. $f(x) \subseteq \bigcap_{a \in c(x)} \delta_f(a)$ for each $x \in X$. Here $\delta_f : \Sigma_0 + \Sigma_2 \times X \times X \to \mathcal{P}Q$ is defined by 1) $\delta_f(s) = \delta(s)$ for $s \in \Sigma_0$; 2) $\delta_f(s, x_1, x_2) = \bigcup_{q_1 \in f(x_1), q_2 \in f(x_2)} \delta(s, q_1, q_2)$ for $s \in \Sigma_2$.
2. $q_{\mathsf{F}} \in f(x_0)$.

▶ **Theorem 7.7.** *In the setting of Def. 7.6, assume that there exists a model checking invariant $f : X \to \mathcal{P}Q$. Then, $\mathcal{A}$ accepts every finite Σ-tree $t \in L_{\mathcal{C}}^{\mathrm{fin}}$ generated by $\mathcal{C}$.*

The proof is in [21, Appendix C.13]. The nondeterminism on the system side ($\mathcal{C}$ in Def. 7.5) requires to work in the effectful setting (§5.2). Another challenge is that the relevant functor lifting is not fibred (cf. the last item in Def. 5.10); we use the coincidence lifting (Thm. 3.6) to deal with it, where the required base coincidence comes from coalgebraic trace semantics [15].

8 Conclusions and Future Work

We presented our IF/I coincidence, which is a general categorical framework for the coincidence of initial algebras and final coalgebras, a classic topic in computer science. The IF/I coincidence is formulated in fibrational terms, and this occurs in the fiber over an initial algebra; it is therefore understood as the coincidence of logical lfp and gfp specifications. Relying on mild assumptions of fibred liftings and stable chain colimits, the IF/I coincidence accommodates many examples. As applications, we derived seemingly new verification methods for probabilistic liveness and tree automata.

The proofs in §6–7 suggest the possibility of a structural theory of the IA-FC coincidence, where unique fixed points are pulled back along coalgebra homomorphisms. We will pursue this structural theory, together with its practical consequences.

Another direction of future work is to formalize the relationship between the current fibrational approach to the IA-FC coincidence, and the homset enrichment approach in [5, 11, 12, 15, 27].

References

1 Jiří Adámek. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae*, 15(4):589–602, 1974.

2 Jiří Adámek, J Adamek, J Rosicky, et al. *Locally presentable and accessible categories*, volume 189. Cambridge University Press, 1994.

3 Jirí Adámek, Stefan Milius, and Lawrence S. Moss. Fixed points of functors. *J. Log. Algebraic Methods Program.*, 95:41–81, 2018. doi:10.1016/j.jlamp.2017.11.003.

4 Alejandro Aguirre and Shin-ya Katsumata. Weakest preconditions in fibrations. *Electronic Notes in Theoretical Computer Science*, 352:5–27, 2020. The 36th Mathematical Foundations of Programming Semantics Conference, 2020. doi:10.1016/j.entcs.2020.09.002.

5 Michael Barr. Algebraically compact functors. *Journal of Pure and Applied Algebra*, 82(3):211–231, 1992.

6 Richard S. Bird and Oege de Moor. *Algebra of programming.* Prentice Hall International series in computer science. Prentice Hall, 1997.

7 Filippo Bonchi, Barbara König, and Daniela Petrisan. Up-to techniques for behavioural metrics via fibrations. *CoRR*, abs/1806.11064, 2018. arXiv:1806.11064.

8 Venanzio Capretta, Tarmo Uustalu, and Varmo Vene. Corecursive algebras: A study of general structured corecursion. In Marcel Vinícius Medeiros Oliveira and Jim Woodcock, editors, *Formal Methods: Foundations and Applications, 12th Brazilian Symposium on Formal Methods, SBMF 2009, Gramado, Brazil, August 19-21, 2009, Revised Selected Papers*, volume 5902 of *Lecture Notes in Computer Science*, pages 84–100. Springer, 2009. doi:10.1007/978-3-642-10452-7_7.

9 Aurelio Carboni, Stephen Lack, and R.F.C. Walters. Introduction to extensive and distributive categories. *Journal of Pure and Applied Algebra*, 84(2):145–158, 1993. doi:10.1016/0022-4049(93)90035-R.

10 Aleksandar Chakarov and Sriram Sankaranarayanan. Probabilistic program analysis with martingales. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 511–526. Springer, 2013. doi:10.1007/978-3-642-39799-8_34.

11 Marcelo P. Fiore. *Axiomatic Domain Theory in Categories of Partial Maps.* Distinguished Dissertations in Computer Science. Cambridge University Press, 1996. doi:10.1017/CBO9780511526565.

12 Peter Freyd. Algebraically complete categories. In Aurelio Carboni, Maria Cristina Pedicchio, and Guiseppe Rosolini, editors, *Category Theory*, pages 95–104, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

13 Peter J. Freyd. Recursive types reduced to inductive types. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science (LICS '90), Philadelphia, Pennsylvania, USA, June 4-7, 1990*, pages 498–507. IEEE Computer Society, 1990. doi:10.1109/LICS.1990.113772.

14 Ichiro Hasuo, Kenta Cho, Toshiki Kataoka, and Bart Jacobs. Coinductive predicates and final sequences in a fibration. In Dexter Kozen and Michael W. Mislove, editors, *Proceedings of the Twenty-ninth Conference on the Mathematical Foundations of Programming Semantics, MFPS 2013, New Orleans, LA, USA, June 23-25, 2013*, volume 298 of *Electronic Notes in Theoretical Computer Science*, pages 197–214. Elsevier, 2013. doi:10.1016/j.entcs.2013.09.014.

15 Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic trace semantics via coinduction. *Log. Methods Comput. Sci.*, 3(4), 2007. doi:10.2168/LMCS-3(4:11)2007.

16 Tobias Heindel and Pawel Sobocinski. Van kampen colimits as bicolimits in span. In Alexander Kurz, Marina Lenisa, and Andrzej Tarlecki, editors, *Algebra and Coalgebra in Computer Science, Third International Conference, CALCO 2009, Udine, Italy, September 7-10, 2009. Proceedings*, volume 5728 of *Lecture Notes in Computer Science*, pages 335–349. Springer, 2009. doi:10.1007/978-3-642-03741-2_23.

17 Horst Herrlich. Topological functors. *General Topology and its Applications*, 4(2):125–142, 1974.

18 Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*, volume 59 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2016. doi:10.1017/CBO9781316823187.

19 Bart P. Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in logic and the foundations of mathematics*. North-Holland, 2001. URL: http://www.elsevierdirect.com/product.jsp?isbn=9780444508539.

20 Yuichi Komorida, Shin-ya Katsumata, Nick Hu, Bartek Klin, and Ichiro Hasuo. Codensity games for bisimilarity. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785691.

21 Mayuko Kori, Ichiro Hasuo, and Shin ya Katsumata. Fibrational initial algebra-final coalgebra coincidence over initial algebras: Turning verification witnesses upside down, 2021. An extended version with appendices. arXiv:2105.04817.

22 Jacob Lurie. *Higher Topos Theory (AM-170)*. Princeton University Press, 2009. URL: http://www.jstor.org/stable/j.ctt7s47v.

23 E. Moggi. Notions of computation and monads. *Inf. & Comp.*, 93(1):55–92, 1991.

24 Philip S. Mulry. Lifting theorems for kleisli categories. In Stephen D. Brookes, Michael G. Main, Austin Melton, Michael W. Mislove, and David A. Schmidt, editors, *Mathematical Foundations of Programming Semantics, 9th International Conference, New Orleans, LA, USA, April 7-10, 1993, Proceedings*, volume 802 of *Lecture Notes in Computer Science*, pages 304–319. Springer, 1993. doi:10.1007/3-540-58027-1_15.

25 Andrew M. Pitts. Relational properties of domains. *Inf. Comput.*, 127(2):66–90, 1996. doi:10.1006/inco.1996.0052.

26 J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000.

27 Michael B. Smyth and Gordon D. Plotkin. The category-theoretic solution of recursive domain equations. *SIAM J. Comput.*, 11(4):761–783, 1982. doi:10.1137/0211062.

28 David Sprunger, Shin-ya Katsumata, Jérémy Dubut, and Ichiro Hasuo. Fibrational bisimulations and quantitative reasoning. In Corina Cîrstea, editor, *Coalgebraic Methods in Computer Science - 14th IFIP WG 1.3 International Workshop, CMCS 2018, Colocated with ETAPS 2018, Thessaloniki, Greece, April 14-15, 2018, Revised Selected Papers*, volume 11202 of *Lecture Notes in Computer Science*, pages 190–213. Springer, 2018. doi:10.1007/978-3-030-00389-0_11.

29 Toru Takisaka, Yuichiro Oyabu, Natsuki Urabe, and Ichiro Hasuo. Ranking and repulsing supermartingales for reachability in probabilistic programs. In *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, pages 476–493, 2018. doi:10.1007/978-3-030-01090-4_28.

30 Věra Trnková, Jiří Adámek, Václav Koubek, and Jan Reiterman. Free algebras, input processes and free monads. *Commentationes Mathematicae Universitatis Carolinae*, 16(2):339–351, 1975.

31 Natsuki Urabe, Masaki Hara, and Ichiro Hasuo. Categorical liveness checking by corecursive algebras. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005151.

32 Vladimir Zamdzhiev. Reflecting algebraically compact functors. In John Baez and Bob Coecke, editors, *Proceedings Applied Category Theory 2019, ACT 2019, University of Oxford, UK, 15-19 July 2019*, volume 323 of *EPTCS*, pages 15–23, 2019. doi:10.4204/EPTCS.323.2.

A IF/I Coincidence for general fibrations

Here we show the following theorem similar to Thm. 4.4 for general fibrations.

▶ **Theorem A.1** (IF/I coincidence for general fibrations). *Let $p : \mathbb{E} \to \mathbb{B}$ be a fibration; assume that $\mathbb{E}$ and $\mathbb{B}$ are chain-cocomplete. Let $\dot{F}$ be a lifting of F along p. Assume further the following conditions:*
1. *The initial chain of $\dot{F}$ converges.*
2. *$\dot{F}$ is a fibred lifting of F.*
3. *p has stable chain colimits.*
4. *p strictly preserves chain colimits.*
5. *Substitution in p preserves chain colimits in fibers.*

Then $(p, F, \dot{F})$ satisfies the IF/I coincidence.

The theorem follows from the next technical observation.

▶ **Proposition A.2.** *In the setting of Thm. A.1, assume further that the initial chain of $\dot{F}$ converges in λ steps. Consider the initial chains:*

$$\begin{tikzcd}
\mathbb{E} \arrow[d, "p"'] & 0 \arrow[r, "\dot{\alpha}_{0,1}"] & \dot{F}0 \arrow[r, "\dot{\alpha}_{1,2}"] & \cdots \arrow[r] & \dot{F}^{\lambda}0 \arrow[r, "\dot{\alpha}_{\lambda,\lambda+1}", "\cong"'] & \dot{F}^{\lambda+1}0 \arrow[r] & \cdots \\
\mathbb{B} & 0 \arrow[r, "\alpha_{0,1}"] & F0 \arrow[r, "\alpha_{1,2}"] & \cdots \arrow[r] & F^{\lambda}0 \arrow[r, "\alpha_{\lambda,\lambda+1}", "\cong"'] & F^{\lambda+1}0 \arrow[r] & \cdots
\end{tikzcd}$$

where 0 in the two lines denote initial objects in $\mathbb{E}$ and in $\mathbb{B}$, respectively. Note that both $\dot{\alpha}_{\lambda,\lambda+1}$ and $\alpha_{\lambda,\lambda+1}$ are isomorphisms: the former is by the convergence assumption; the latter is by the assumption and Lem. 2.10. Therefore their inverses are initial algebras by Prop. 2.8.

In this setting, $\dot{\alpha}_{\lambda,\lambda+1}$ is a final object of $\mathbf{Coalg}(\dot{F})_{\alpha_{\lambda,\lambda+1}}$ *(cf. Def. 3.1).*

From now on, we aim to prove Prop. A.2.

To show finality of $\dot{\alpha}_{\lambda,\lambda+1}$ in $\mathbf{Coalg}(\dot{F})_{\alpha_{\lambda,\lambda+1}}$, we first claim the existence of a morphism from an arbitrary $\dot{F}$-coalgebra $\gamma : P \to \dot{F}P$ in $\mathbf{Coalg}(\dot{F})_{\alpha_{\lambda,\lambda+1}}$ to $\dot{\alpha}_{\lambda,\lambda+1}$. The next lemma shows a construction of such a morphism p_λ by transfinite induction along initial chains.

$$\begin{tikzcd}
\mathbb{E} \arrow[dd, "p"] & \dot{F}^{\lambda}0 \arrow[r, "\dot{\alpha}_{\lambda,\lambda+1}"] & \dot{F}^{\lambda+1}0 \\
& P \arrow[u, "p_\lambda"] \arrow[r, "\gamma"] & \dot{F}P \arrow[u, "\dot{F}p_\lambda"'] \\
\mathbb{B} & F^{\lambda}0 \arrow[r, "\alpha_{\lambda,\lambda+1}"] & F^{\lambda+1}0
\end{tikzcd}$$

This construction exploits the singleton property of $\mathbb{E}_0$ (Lem. 4.2) in the base case, fibredness of the lifting $\dot{F}$ in the step case, and stability of chain colimits in the limit case.

▶ **Lemma A.3.** *Let $p : \mathbb{E} \to \mathbb{B}$ be a fibration with stable chain colimits. Assume that $\mathbb{E}$ and $\mathbb{B}$ are chain-cocomplete and p strictly preserves chain colimits. Let $\dot{F}$ be a fibred lifting of F along p.*

For each ordinal λ and each coalgebra $\gamma : P \to \dot{F}P$ above $\alpha_{\lambda,\lambda+1}$ (or equivalently, $\gamma \in$ $\mathbf{Coalg}(\dot{F})_{\alpha_{\lambda,\lambda+1}}$*), there exists a morphism $p_\lambda : \gamma \to \dot{\alpha}_{\lambda,\lambda+1}$ in* $\mathbf{Coalg}(\dot{F})_{\alpha_{\lambda,\lambda+1}}$*.*

Proof. Let $(k_{j,i} : \alpha^*_{j,\lambda}P \to \alpha^*_{i,\lambda}P)_{j \le i \le \lambda}$ denote those morphisms induced from $(\alpha^*_{i,\lambda}P \to P)_{i \le \lambda}$ via their universality as cartesian liftings. We construct vertical morphisms $(p_i : \alpha^*_{i,\lambda}P \to \dot{F}^i 0)_{i \le \lambda}$ such that $p_i \circ k_{j,i} = \dot{\alpha}_{j,i} \circ p_j$ for all i, j with $j \le i \le \lambda$. Such a $(p_i)_{i \le \lambda}$ makes the diagram below commute. (The rightmost square commutes, see [21, Appendix C.14] for details.)

$$\begin{tikzcd}
\mathbb{E} \arrow[dd, "p"'] & 0 \arrow[r, "\dot\alpha_{0,1}"] & \dot F 0 \arrow[r, "\dot\alpha_{1,2}"] & \cdots \arrow[r] & \dot F^{\lambda} 0 \arrow[r, "\dot\alpha_{\lambda,\lambda+1}"] & \dot F^{\lambda+1} 0 \\
 & \alpha^*_{0,\lambda}P \arrow[u, "p_0"] \arrow[r, "k_{0,1}"] & \alpha^*_{1,\lambda}P \arrow[u, "p_1"] \arrow[r, "k_{1,2}"] & \cdots \arrow[r] & \alpha^*_{\lambda,\lambda}P = P \arrow[u, "p_\lambda"] \arrow[r, "\gamma"] & \dot F P \arrow[u, "\dot F p_\lambda"] \\
\mathbb{B} & 0 \arrow[r, "\alpha_{0,1}"] & F 0 \arrow[r, "\alpha_{1,2}"] & \cdots \arrow[r] & F^{\lambda} 0 \arrow[r, "\alpha_{\lambda,\lambda+1}"] & F^{\lambda+1} 0
\end{tikzcd}$$

Then this p_λ is what we want. The construction of $(p_i)_{i\leq\lambda}$ is by the following transfinite induction on i.

- (Base case) Lem. 4.2 says $\alpha^*_{0,\lambda}P \cong 0$ in $\mathbb{E}$ and this isomorphism is vertical because both $p\alpha^*_{0,\lambda}P$ and $p0$ are 0 in $\mathbb{B}$. We define p_0 as this isomorphism.
- (Step case) If i is a successor ordinal, we define p_i by

$$\alpha^*_{i,\lambda}P \xrightarrow{(\star)} (F\alpha_{i-1,\lambda})^*\dot F P \underset{\cong}{\xrightarrow{\xi}} \dot F \alpha^*_{i-1,\lambda}P \xrightarrow{\dot F p_{i-1}} \dot F^i 0$$

where ξ is from fibredness of $\dot F$ and $(\star)$ is induced as follows by universality of a cartesian lifting.

$$\begin{tikzcd}
 & \alpha^*_{i,\lambda}P \arrow[r, "\overline{\alpha_{i,\lambda}}"'] \arrow[d, dotted, "(\star)"] & P \arrow[dr, bend left, "\gamma"] & \\
 & (F\alpha_{i-1,\lambda})^*\dot F P \arrow[rr, "\overline{F\alpha_{i-1,\lambda}}"] & & \dot F P \\
 & F^i 0 \arrow[r, "\alpha_{i,\lambda}"] & F^{\lambda} 0 \arrow[r, "\alpha_{\lambda,\lambda+1}"] & F^{\lambda+1} 0.
\end{tikzcd}$$

Note that $F\alpha_{i-1,\lambda} = \alpha_{\lambda,\lambda+1} \circ \alpha_{i,\lambda}$ by the definition of α. We can prove $\dot\alpha_{j,i} \circ p_j = p_i \circ k_{j,i}$ for all $j \leq i$ by transfinite induction on j. See [21, Appendix C.14].

- (Limit case) If i is a limit ordinal, we define p_i by the stability of chain colimits. By applying chain colimit stability of p to $\alpha^*_{i,\lambda}P$ above $F^i 0 = \mathrm{colim}_{j<i} F^j 0$ (by Def. 2.7, see below), we have $\alpha^*_{i,\lambda}P \cong \mathrm{colim}_{j<i}\, \alpha^*_{j,i}(\alpha^*_{i,\lambda}P) \cong \mathrm{colim}_{j<i}\, \alpha^*_{j,\lambda}P$. For all l, j with $l < j < i$, by the induction hypothesis, we have $\dot\alpha_{l,i} \circ p_l = \dot\alpha_{j,i} \circ \dot\alpha_{l,j} \circ p_l = \dot\alpha_{j,i} \circ p_j \circ k_{l,j}$. Hence $(\dot F^i 0, (\alpha_{j,i} \circ p_j)_{j<i})$ is a cocone over $(j \mapsto \alpha^*_{j,\lambda}P)$, as shown below. We define p_i as the mediating morphism from a colimit, as in the following diagram.

$$\begin{tikzcd}
 & & & \dot F^i 0 \text{ (colim.)} & \\
\cdots \arrow[r] & \dot F^l 0 \arrow[r] \arrow[urr, bend left, "\dot\alpha_{l,i}"] & \dot F^j 0 \arrow[r] \arrow[ur, bend left, "\dot\alpha_{j,i}"'] & \cdots & \\
\cdots \arrow[r] & \alpha^*_{l,\lambda}P \arrow[u, "p_l"] \arrow[r] \arrow[drr, bend right, "k_{l,i}"'] & \alpha^*_{j,\lambda}P \arrow[u, "p_j"] \arrow[r] \arrow[dr, bend right, "k_{j,i}"] & \cdots & \\
 & & & \alpha^*_{i,\lambda}P \text{ (colim.)} \arrow[uuu, dotted, bend right, "p_i"'] & \\
\cdots \arrow[r] & F^l 0 \arrow[r, "\alpha_{l,j}"] & F^j 0 \arrow[r, "\alpha_{j,i}"] & F^i 0 \arrow[r] & \cdots
\end{tikzcd}$$

This concludes the proof. ◀

For Prop. A.2, it remains to show the uniqueness of $p_\lambda : \gamma \to \dot\alpha_{\lambda,\lambda+1}$. The uniqueness does not immediately follow from the construction of p_λ in Lem. A.3.

Our uniqueness proof (in the proof of Prop. A.2 shown later), we work on a suitable chain in the fiber $\mathbb{E}_{F^\lambda 0}$, defined as follows.

The following fact (cf. [19, Prop. 9.2.2 and Exercise 9.2.4]) shows $\mathbf{CLat}_\wedge$-fibrations have properties suitable for colimits.

▶ **Proposition A.4.** *Let $p : \mathbb{E} \to \mathbb{B}$ be an opfibration. Assume the base category $\mathbb{B}$ has colimits of shape $\mathbb{I}$. Then the following statements are equivalent.*

1. *Each fiber of the opfibration p has colimits of shape $\mathbb{I}$.*
2. *The total category $\mathbb{E}$ has colimits of shape $\mathbb{I}$ and p strictly preserves them.*

▶ **Notation A.5** **($\dot\alpha, \alpha$).** In the setting of Lem. A.3, let us fix λ to be a converging ordinal of the initial $\dot F$-chain, in the sense that $\dot\alpha_{\lambda,\lambda+1}\colon \dot F^\lambda 0 \xRightarrow{\cong} \dot F^{\lambda+1}0$ is an isomorphism. In the rest of the section, we write $\dot\alpha, \alpha$ for $\dot\alpha_{\lambda,\lambda+1}, \alpha_{\lambda,\lambda+1}$, respectively. Then $\alpha\colon F^\lambda 0 \xRightarrow{\cong} F^{\lambda+1}0$ is an isomorphism, too.

▶ **Definition A.6.** In Prop. A.2, we define a chain

$$P \xrightarrow{\beta_{0,1}} \alpha^*\dot FP \xrightarrow{\beta_{1,2}} (\alpha^*\dot F)^2P \xrightarrow{\beta_{2,3}} \cdots$$

by repeated application of $\alpha^*\dot F$. The whole chain resides in the fiber $\mathbb{E}_{F^\lambda 0}$, as shown in (4).

$$\begin{tikzcd}
\vdots & \vdots \\
(\alpha^*\dot F)^2P \arrow[u] \arrow[rd, "\bar\alpha"] & \dot F(\alpha^*\dot F)^2P \arrow[u] \\
\alpha^*\dot FP \arrow[u, "\beta_{1,2}"] \arrow[rd, "\bar\alpha"] & \dot F\alpha^*\dot FP \arrow[u, "\dot F\beta_{1,2}"'] \\
P \arrow[u, "\beta_{0,1}"] \arrow[r, "\gamma"'] & \dot FP \arrow[u, "\dot F\beta_{0,1}"'] \\
F^\lambda 0 \arrow[r, "\alpha"] & F^{\lambda+1}0
\end{tikzcd} \tag{4}$$

The precise definition is as follows. It is similar to Def. 2.7, but starting from P (instead of from 0) calls for some care.

- (Objects) $(\alpha^*\dot F)^iP$ is given for each $i \in \mathrm{Ord}$: $(\alpha^*\dot F)^0P = P$, $(\alpha^*\dot F)^{i+1}P = \alpha^*\dot F((\alpha^*\dot F)^iP)$, and $(\alpha^*\dot F)^i0 = \mathrm{colim}_{j<i}(\alpha^*\dot F)^j0$ for a limit ordinal i.
- (Morphisms) The morphism $\beta_{i,i+1}\colon (\alpha^*\dot F)^i0 \to (\alpha^*\dot F)^{i+1}0$ for each ordinal i is defined as follows.
  - (Base case) $\beta_{0,1} : P \to \alpha^*\dot FP$ is induced from $\gamma\colon P \to \dot FP$ by universality of the cartesian lifting $\bar\alpha : \alpha^*\dot FP \to \dot FP$. See (4).
  - (Step case) $\beta_{i+1,i+2}$ is defined by $\alpha^*\dot F\beta_{i,i+1}$.
  - (Limit case) $\beta_{i,i+1} : (\alpha^*\dot F)^iP \to (\alpha^*\dot F)^{i+1}P$ for a limit ordinal i is induced by universality of $(\alpha^*\dot F)^iP = \mathrm{colim}_{j<i}(\alpha^*\dot F)^jP$. Prop. A.4 ensures this colimit vertex is above $F^\lambda 0$.

We have defined $\beta_{j,j+1}$ for each ordinal j. This induces morphisms $\beta_{i,j}\colon (\alpha^*\dot F)^iP \to (\alpha^*\dot F)^jP$ for each $i < j$ in a straight-forward manner: one repeats the step and limit cases; when j is a limit ordinal, $\beta_{i,j}$ is the cocone component to $(\alpha^*\dot F)^jP = \mathrm{colim}_{k<j}(\alpha^*\dot F)^kP$.

▶ **Lemma A.7.** *In the setting of Prop. A.2, let us assume that λ is a converging ordinal in the initial chain of $\dot F$, and adopt Notation A.5.*

Fig. 2 shows the following constructs.

- *The morphism $\gamma : P \to \dot FP$ above α induces the chain $P \xrightarrow{\beta_{0,1}} \alpha^*\dot FP \xrightarrow{\beta_{1,2}} (\alpha^*\dot F)^2P \to \cdots$ as in Def. A.6.*
- *The last chain induces, for each ordinal l such that $l < \lambda$, the chain*

  $$\alpha^*_{l,\lambda}P \xrightarrow{\alpha^*_{l,\lambda}\beta_{0,1}} \alpha^*_{l,\lambda}\alpha^*\dot FP \xrightarrow{\alpha^*_{l,\lambda}\beta_{1,2}} \alpha^*_{l,\lambda}(\alpha^*\dot F)^2P \to \cdots$$

  above F^l0, via the substitution along $\alpha_{l,\lambda}$.

- *For each ordinal i, we obtain a $\dot{F}$-coalgebra as follows. It is above α; it is denoted by γ_i.*

$$\begin{tikzcd}
(\alpha^*\dot{F})^iP \arrow[r, "\gamma_i := \overline{\alpha}\circ\beta_{i,i+1}"] & \dot{F}(\alpha^*\dot{F})^iP \\
F^\lambda 0 \arrow[r, "\alpha", "\cong"'] & F^{\lambda+1}0
\end{tikzcd} \tag{5}$$

- *We apply Lem. A.3 to the last coalgebras γ_i, using each of them in place of the coalgebra γ in Lem. A.3. Following the proof of Lem. A.3, we obtain vertical morphisms $(p^i_j : \alpha^*_{j,\lambda}(\alpha^*\dot{F})^iP \to \dot{F}^j0)_{j\leq\lambda}$ for each ordinal i.*

In this case, for each i such that $i \leq \lambda$, (i) p^i_i is an isomorphism; (ii) $p^l_i = p^m_i \circ \alpha^*_{i,\lambda}\beta_{l,m}$ for all l, m with $l \leq m$.

See [21, Appendix C.15] for the proof.

$$\begin{tikzcd}
 & 0 \arrow[r] & \dot{F}0 \arrow[r] & \dot{F}^2 0 \arrow[r] & \cdots \arrow[r] & \dot{F}^\lambda 0 \arrow[r, "\dot{\alpha}"] & \dot{F}^{\lambda+1}0 \\
 & \vdots & \vdots & \vdots & & \vdots & \vdots \\
\mathbb{E} \arrow[ddd, "p"] & \alpha^*_{0,\lambda}(\alpha^*\dot{F})^2P \arrow[r] \arrow[u] \arrow[uu, bend left, "p^2_0"] & \alpha^*_{1,\lambda}(\alpha^*\dot{F})^2P \arrow[r] \arrow[u] \arrow[uu, bend left, "p^2_1"] & \alpha^*_{2,\lambda}(\alpha^*\dot{F})^2P \arrow[r] \arrow[u] \arrow[uu, bend left, "p^2_2"] & \cdots \arrow[r] & (\alpha^*\dot{F})^2P \arrow[u] \arrow[uu, bend left, "p^2_\lambda"] \arrow[dr, "\overline{\alpha}"] & \dot{F}(\alpha^*\dot{F})^2P \arrow[u] \arrow[uu, bend right, "\dot{F}p^2_\lambda"'] \\
 & \alpha^*_{0,\lambda}\alpha^*\dot{F}P \arrow[r] \arrow[u, "\alpha^*_{0,\lambda}\beta_{1,2}"'] \arrow[uuu, bend left, "p^1_0"] & \alpha^*_{1,\lambda}\alpha^*\dot{F}P \arrow[r] \arrow[u, "\alpha^*_{1,\lambda}\beta_{1,2}"'] \arrow[uuu, bend left, "p^1_1"] & \alpha^*_{2,\lambda}\alpha^*\dot{F}P \arrow[r] \arrow[u, "\alpha^*_{2,\lambda}\beta_{1,2}"'] \arrow[uuu, bend left, "p^1_2"] & \cdots \arrow[r] & \alpha^*\dot{F}P \arrow[u, "\beta_{1,2}"] \arrow[uuu, bend left, "p^1_\lambda"] \arrow[dr, "\overline{\alpha}"] & \dot{F}\alpha^*\dot{F}P \arrow[u, "\dot{F}\beta_{1,2}"] \arrow[uuu, bend right, "\dot{F}p^1_\lambda"'] \\
 & \alpha^*_{0,\lambda}P \arrow[r] \arrow[u, "\alpha^*_{0,\lambda}\beta_{0,1}"'] \arrow[uuuu, bend left, "p^0_0"] & \alpha^*_{1,\lambda}P \arrow[r] \arrow[u, "\alpha^*_{1,\lambda}\beta_{0,1}"'] \arrow[uuuu, bend left, "p^0_1"] & \alpha^*_{2,\lambda}P \arrow[r] \arrow[u, "\alpha^*_{2,\lambda}\beta_{0,1}"'] \arrow[uuuu, bend left, "p^0_2"] & \cdots \arrow[r] & P \arrow[r, "\gamma"] \arrow[u, "\beta_{0,1}"] \arrow[uuuu, bend left, "p^0_\lambda"] & \dot{F}P \arrow[u, "\dot{F}\beta_{0,1}"] \arrow[uuuu, bend right, "\dot{F}p^0_\lambda"'] \\
\mathbb{B} & 0 \arrow[r] & F0 \arrow[r] & F^2 0 \arrow[r] & \cdots \arrow[r] & F^\lambda 0 \arrow[r, "\alpha"] & F^{\lambda+1}0
\end{tikzcd}$$

Figure 2 A diagram for Lem. A.7.

The last lemma shows that the $\dot{F}$-coalgebra $\gamma_i : (\alpha^*\dot{F})^iP \to \dot{F}(\alpha^*\dot{F})^iP$ gets closer to $\dot{\alpha}$ as i gets larger, with a particular consequence that γ_λ is isomorphic to $\dot{\alpha}$ (via p^λ_λ). This is used in the following proof of Prop. A.2.

Proof of Prop. A.2. Let γ be an arbitrary coalgebra $P \to \dot{F}P$ above $\alpha = \alpha_{\lambda,\lambda+1}$ (Notation A.5). Lem. A.3 shows the existence of a vertical morphism from γ to $\dot{\alpha}_{\lambda,\lambda+1}$. We only need to show the uniqueness of morphisms. Let f be an arbitrary vertical morphism from γ to $\dot{\alpha}_{\lambda,\lambda+1}$. The isomorphic correspondence in Lem. 3.4 carries $f : (P \xrightarrow{\gamma} \dot{F}P) \to (\dot{F}^\lambda 0 \xrightarrow{\dot{\alpha}} \dot{F}^{\lambda+1}0)$ in $\mathbf{Coalg}(\dot{F})_\alpha$ to $f : (P \xrightarrow{\beta_{0,1}} \alpha^*\dot{F}P) \to (\dot{F}^\lambda 0 \xrightarrow{\delta} \alpha^*\dot{F}^{\lambda+1}0)$ in $\mathbf{Coalg}(\alpha^*\dot{F})$, where δ is the mediating morphism from $\dot{\alpha}$ by universality of the cartesian lifting $\overline{\alpha} : \alpha^*\dot{F}^{\lambda+1}0 \to \dot{F}^{\lambda+1}0$.

Using the above f in $\mathbf{Coalg}(\alpha^*\dot{F})$, we consider the following two chains and a morphism between them. Everything here is above $F^\lambda 0$; cf. (4). δ is an isomorphism since the initial chain of $\dot{F}$ converges in λ steps.

$$\begin{tikzcd}
\dot{F}^\lambda 0 \arrow[r, "\cong", "\delta"'] & (\alpha^*\dot{F})\dot{F}^\lambda 0 \arrow[r, "\cong"] & \cdots \arrow[r, "\cong"] & (\alpha^*\dot{F})^\lambda\dot{F}^\lambda 0 \arrow[r, "\cong"] & (\alpha^*\dot{F})^{\lambda+1}\dot{F}^\lambda 0 \\
P \arrow[u, "f"] \arrow[r, "\beta_{0,1}"'] & (\alpha^*\dot{F})P \arrow[u, "(\alpha^*\dot{F})f"'] \arrow[r] & \cdots \arrow[r, "\beta_{\lambda-1,\lambda}"'] & (\alpha^*\dot{F})^\lambda P \arrow[u, "(\alpha^*\dot{F})^\lambda f"] \arrow[r, "\beta_{\lambda,\lambda+1}"'] & (\alpha^*\dot{F})^{\lambda+1}P \arrow[u, "(\alpha^*\dot{F})^{\lambda+1}f"]
\end{tikzcd} \tag{6}$$

It follows easily that $\beta_{\lambda,\lambda+1}$ is the inverse of an initial $\alpha^*\dot{F}$-algebra. This is essentially because 1) γ_λ is isomorphic to $\dot{\alpha}$ (see the paragraph that follows Lem. A.7); and 2) (the inverse of) $\gamma_\lambda = \overline{\alpha} \circ \beta_{\lambda,\lambda+1}$ corresponds to (the inverse of) $\beta_{\lambda,\lambda+1}$ in the isomorphic correspondence $\mathbf{Alg}(\dot{F})_{\alpha^{-1}} \cong \mathbf{Alg}(\alpha^*\dot{F})$ in Lem. 3.4.

Consider the rightmost square in (6). By universality of the initial $\alpha^*\dot{F}$-algebra $(\beta_{\lambda,\lambda+1})^{-1}$, $(\alpha^*\dot{F})^\lambda f$ is unique; therefore the composite $(\alpha^*\dot{F})^\lambda f \circ \beta_{\lambda-1,\lambda} \circ \cdots \circ \beta_{0,1}$ (on the left in (6)) is uniquely determined. By the commutativity of (6) and the fact that all the morphisms in the first row are isomorphisms, this uniquely determines f, too. ◀

SMT-Based Model Checking of Max-Plus Linear Systems

Muhammad Syifa'ul Mufid
Department Computer Science, University of Oxford, UK

Andrea Micheli
Fondazione Bruno Kessler, Trento, Italy

Alessandro Abate
Department Computer Science, University of Oxford, UK

Alessandro Cimatti
Fondazione Bruno Kessler, Trento, Italy

Abstract

Max-Plus Linear (MPL) systems are an algebraic formalism with practical applications in transportation networks, manufacturing and biological systems. MPL systems can be naturally modeled as infinite-state transition systems, and exhibit interesting structural properties (e.g. periodicity or steady state), for which analysis methods have been recently proposed. In this paper, we tackle the open problem of specifying and analyzing user-defined temporal properties for MPL systems. We propose Time-Difference LTL (TDLTL), a logic that encompasses the delays between the discrete-time events governed by an MPL system, and characterize the problem of model checking TDLTL over MPL. We propose a family of specialized algorithms leveraging the periodic behaviour of an MPL system. We prove soundness and completeness, showing that the transient and cyclicity of the MPL system induce a completeness threshold for the verification problem. The algorithms are cast in the setting of SMT-based verification of infinite-state transition systems over the reals, with variants depending on the (incremental vs upfront) computation of the bound, and on the (explicit vs implicit) unrolling of the transition relation. Our comprehensive experiments show that the proposed techniques can be applied to MPL systems of large dimensions and on general TDLTL formulae, with remarkable performance gains against a dedicated abstraction-based technique and a translation to the nuXmv symbolic model checker.

2012 ACM Subject Classification Theory of computation → Verification by model checking

Keywords and phrases Max-Plus Linear Systems, Satisfiability Modulo Theory, Model Checking, Linear Temporal Logic

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.22

1 Introduction

Max-Plus Linear (MPL) systems are a class of discrete-event systems (DES) that are based on the so-called max-plus algebra, an algebraic system using the two binary operations of maximisation and addition. MPL systems are employed to model applications with features of synchronization without concurrency, and as such are widely used for applications in transportation networks [7], manufacturing [29] and biological systems [14, 20]. In MPL models, the states correspond to time instances related to discrete events. Traditional dynamical analysis of MPL systems is associated with their algebraic and graph representation (cf. Definition 2), that allows the investigation of several structural problems such as eigenproblems [21], optimisation [13] and periodicity [24, 35].

In this paper, we tackle the problem of formally specifying and analyzing user-defined temporal properties for MPL systems. This can be considered an open problem in practice, despite the line of work in [3, 4, 5] that deals with reachability analysis and formal verification

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 22; pp. 22:1–22:20
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

for MPL systems. These methods are in general not complete: they rely on the construction of an abstraction that overapproximates the concrete MPL system [2, 33, 34]. Furthermore, the underlying abstraction procedures suffer from state-explosion problems, given that the size of the abstraction is exponential in the size of the MPL, and are unable to deal with more than few variables. Finally, no general specification language is provided to express properties at the MPL system level.

We make the following contributions. First, we propose Time-Difference LTL (TDLTL), a logic that encompasses the delays between the discrete-time events governed by an MPL system, and characterize the problem of model checking TDLTL over MPL. Second, we propose a family of specialized algorithms for TDLTL model checking, cast in the setting of infinite-state transition systems, with a symbolic representation in Satisfiability Modulo Theories (SMT) over the reals [9]. The algorithms, that we prove sound and complete, leverage the periodic behaviour of an MPL system: intuitively, the transient and cyclicity of the MPL system induce a completeness threshold for a bounded encoding of the verification problem. The family of algorithms has several variants, depending on two independent factors. One is the *computation of the bound*, that could be carried out either upfront before calling the SMT solver, or incrementally, interleaving it with multiple solver calls. The other is the *unrolling of the transition relation*, that can either follow the explicit approach of Bounded Model Checking (BMC), or – thanks to some algebraic properties of MPL systems – be left implicit, so that the number of SMT variables is significantly reduced. Third, we demonstrate the practical effectiveness of the approach. We run a comprehensive set of experiments, showing that the proposed techniques can be applied to general TDLTL formulae on large MPL systems that are completely out-of-reach for existing abstraction-based techniques [33, 34]. The comparison also shows that the new algorithms yield orders-of-magnitude speed ups against a generic translational approach to the nuXmv symbolic model checker [15].

The structure is as follows. Section 2 describes the basics of MPL systems and SMT. In Section 3, we formalize the TDLTL logic and Sections 4 and 5 present the verification algorithms and the related work, respectively. Our experimental analysis is reported in Section 6 and we conclude in Section 7. Proofs and additional experiments are provided in the appendices.

2 Model and Preliminaries

2.1 Max-Plus Linear Systems

Max-plus algebra is a modification of the canonical linear algebra, and is defined over the max-plus semiring $(\mathbb{R}_{\max}, \oplus, \otimes)$, where $\mathbb{R}_{\max} := \mathbb{R} \cup \{\varepsilon := -\infty\}$ and

$$a \oplus b := \max\{a, b\}, \qquad a \otimes b := a + b, \quad \forall a, b \in \mathbb{R}_{\max} \tag{1}$$

The zero and unit elements of $\mathbb{R}_{\max}$ are ε and 0, respectively. By $\mathbb{R}^{m\times n}_{\max}$, we denote the set of $m \times n$ matrices over the max-plus algebra. Max-plus algebraic operations can be extended to vectors and matrices as follows. Given $A, B \in \mathbb{R}^{m\times n}_{\max}, C \in \mathbb{R}^{m\times p}_{\max}, D \in \mathbb{R}^{p\times n}_{\max}$ and $\alpha \in \mathbb{R}_{\max}$,

$$\begin{aligned}
&[\alpha \otimes A](i, j) = \alpha + A(i, j),\\
&[A \oplus B](i, j) = A(i, j) \oplus B(i, j),\\
&[C \otimes D](i, j) = \bigoplus_{k=1}^{p} C(i, k) \otimes D(k, j),
\end{aligned}$$

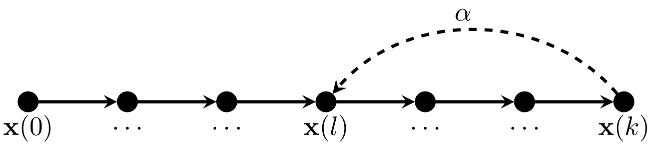
for all $i = 1, \ldots, m$ and $j = 1, \ldots, n$. Given $A \in \mathbb{R}^{n\times n}_{\max}$ and $t \in \mathbb{N}$, $A^{\otimes t}$ denotes $A \otimes \ldots \otimes A$ (t times).

A dynamical system over the max-plus algebra is called a Max-Plus Linear (MPL) system and is defined as

$$\mathbf{x}(k+1) = A \otimes \mathbf{x}(k), \quad k = 0, 1, \dots, \tag{2}$$

where $A \in \mathbb{R}^{n\times n}_{\max}$ is the system matrix, and vector $\mathbf{x}(k) = [x_1(k) \ \dots \ x_n(k)]^\top$ encodes the state variables [7]. Vector $\mathbf{x}$ is used to represent the time stamps associated to the discrete events, while k corresponds to the event counter. Applications of MPL systems are found in dynamical systems where modelling the time variable is essential, such as in transportation networks [29], in scheduling [6] or manufacturing [30] problems, or for biological systems [14, 20].

▶ **Definition 1** (Orbit and Lasso)**.** Given an MPL system (2) with an initial vector $\mathbf{x}(0)$, a sequence $\mathbf{x}(0)\mathbf{x}(1)\dots$ is called an orbit from $\mathbf{x}(0)$ w.r.t. A. Furthermore, if there exist $\alpha \in \mathbb{R}$ and $k \geq l \geq 0$ such that $\mathbf{x}(k+1+j) = \alpha \otimes \mathbf{x}(l+j)$ for $j \geq 0$ then such sequence is called a (k,l)-lasso. Furthermore, we call l the *loopback* bound. The illustration of a lasso is shown in Figure 1.



■ **Figure 1** An illustration of a (k,l)-lasso. The dashed arrow represents the transition $\mathbf{x}(k+1) = \alpha \otimes \mathbf{x}(l)$.

Let us remark that an orbit represents the execution (or path) of (2) originating from an initial state that is a vector. It is important to note that the definition of lasso is slightly different from the *canonical* one found in literature [11, 12], which requires that the l-th and $(k+1)$-th states are the same. The notation $\mathsf{Orb}(A) = \{\mathbf{x}(0)\mathbf{x}(1)\dots \mid \mathbf{x}(0) \in \mathbb{R}^n\}$ represents the set of all orbits w.r.t. A. Likewise, $\mathsf{Orb}(A, X) = \{\mathbf{x}(0)\mathbf{x}(1)\dots \mid \mathbf{x}(0) \in X\}$ is the set of orbits w.r.t. A starting from a set of initial vectors X. For the sake of simplicity we may use the following notation to refer to an orbit: $\pi = \mathbf{x}(0)\mathbf{x}(1)\dots$. Given an orbit π and $j \geq 0$, $\pi[j] = \mathbf{x}(j)$ denotes the j-th vector of π while $\pi[j..]$ is the j-th suffix of π, i.e., $\pi[j..] = \mathbf{x}(j)\mathbf{x}(j+1)\dots$. We say that orbit π is *similar* to orbit σ iff there exists $\beta \in \mathbb{R}$ such that $\pi[0] = \beta \otimes \sigma[0]$ (which implies $\pi[j] = \beta \otimes \sigma[j]$ for $j \geq 0$).

▶ **Definition 2** (Precedence Graph [7])**.** The precedence graph of $A \in \mathbb{R}^{n\times n}_{\max}$, denoted by $\mathcal{G}(A)$, is a weighted directed graph with nodes $1, \dots, n$ and an edge from j to i with weight $A(i,j)$ for each $A(i,j) \neq \varepsilon$.

▶ **Definition 3** (Irreducible Matrix [7])**.** A matrix $A \in \mathbb{R}^{n\times n}_{\max}$ is called *irreducible* if $\mathcal{G}(A)$ is strongly connected.

A directed graph is strongly connected if, for any two different nodes i, j, there exists a path from i to j. The weight of a path $p = i_1 i_2 \dots i_k$ is equal to the sum of the edge weights in p. A circuit, namely a path that begins and ends at the same node, is called *critical* if it has maximum average weight, which is the weight divided by the length of the path [7].

Each irreducible matrix $A \in \mathbb{R}^{n\times n}_{\max}$ admits a unique max-plus eigenvalue $\lambda \in \mathbb{R}$ and a corresponding max-plus eigenspace $E(A) = \{x \in \mathbb{R}^n_{\max} \mid A \otimes x = \lambda \otimes x\}$. The scalar λ is equal to the average weight of critical circuits in $\mathcal{G}(A)$, and $E(A)$ can be computed from

$A_\lambda^+ = \bigoplus_{k=1}^n((-\lambda) \otimes A)^{\otimes k}$. A reducible matrix may have multiple eigenvalues, where the maximum one equals to the average weight of critical circuits of $\mathcal{G}(A)$. Another important property of irreducible MPL systems is the periodicity of the powers of matrix $A^{\otimes k}$.

▶ **Proposition 4** (Transient [7, 29])**.** For an irreducible matrix $A \in \mathbb{R}_{\max}^{n\times n}$ and its max-plus eigenvalue $\lambda \in \mathbb{R}$, there exist $l, c \in \mathbb{N}_0$, such that $A^{\otimes(k+c)} = (\lambda \times c) \otimes A^{\otimes k}$ for all $k \geq l$. The smallest such l and c are called the *transient* and the *cyclicity* of A, respectively.

For the rest of this paper, we denote the transient and the cyclicity of A as $\mathsf{tr}(A)$ and $\mathsf{cyc}(A)$, respectively. While $\mathsf{cyc}(A)$ is related to critical circuits in the precedence graph $\mathcal{G}(A)$ [7, Def 3.94], $\mathsf{tr}(A)$ is unrelated to the dimension of A. Even for a small n, the transient of $A \in \mathbb{R}_{\max}^{n\times n}$ can be large. Upper bounds of the transient have been discussed in [16, 32, 35, 36].

By Proposition 4, all orbits of an irreducible MPL system induce a *periodic* behaviour with a rate λ: for each initial vector $\mathbf{x}(0) \in \mathbb{R}^n$ we have $\mathbf{x}(k + \mathsf{cyc}(A)) = (\lambda \times \mathsf{cyc}(A)) \otimes \mathbf{x}(k)$ for all $k \geq \mathsf{tr}(A)$. A similar condition may be found on reducible MPL systems: we denote the corresponding transient and cyclicity to be global, as per Proposition 4. The local transient and cyclicity for a specific initial vector $\mathbf{x}(0) \in \mathbb{R}^n$ and for a set of initial vectors $X \subseteq \mathbb{R}^n$ has been studied in [1] and is defined as follows.

▶ **Definition 5** ([34])**.** Given $A \in \mathbb{R}_{\max}^{n\times n}$ with max-plus eigenvalue λ and an initial vector $\mathbf{x}(0) \in \mathbb{R}^n$, the local transient and cyclicity of A w.r.t. $\mathbf{x}(0)$ are respectively the smallest $l, c \in \mathbb{N}_0$ such that $\mathbf{x}(j + c) = \lambda c \otimes \mathbf{x}(j), \forall j \geq l$. We denote those scalars as $\mathsf{tr}(A, \mathbf{x}(0))$ and $\mathsf{cyc}(A, \mathbf{x}(0))$, respectively. Furthermore, for $X \subseteq \mathbb{R}^n$, $\mathsf{tr}(A, X) = \max\{\mathsf{tr}(A, \mathbf{x}(0)) \mid \mathbf{x}(0) \in X\}$ and $\mathsf{cyc}(A, X) = \mathtt{lcm}\{\mathsf{cyc}(A, \mathbf{x}(0)) \mid \mathbf{x}(0) \in X\}$, where $\mathtt{lcm}$ is the “least common multiple”.

Following [1], any MPL system (2) can be classified into three categories: 1) *never periodic* if $\mathsf{tr}(A, \mathbf{x}(0))$ does not exist for all $\mathbf{x}(0) \in \mathbb{R}^n$; 2) *boundedly periodic* if $\mathsf{tr}(A, \mathbf{x}(0))$ exists for all $\mathbf{x}(0) \in \mathbb{R}^n$ and $\mathsf{tr}(A)$ exists; and 3) *unboundedly periodic* if $\mathsf{tr}(A, \mathbf{x}(0))$ exists for all $\mathbf{x}(0) \in \mathbb{R}^n$ but $\mathsf{tr}(A)$ does not. We call (2) *periodic* if it is either *unboundedly periodic* or *boundedly periodic*. It has been shown in [1] that an MPL system (2) is periodic if and only if the state matrix A admits a finite eigenvector (all elements are not equal to ε). The following proposition shows the relation between the periodicity of the MPL system (2) and its corresponding orbits.

▶ **Proposition 6.** An orbit π is a lasso iff $\pi[0]$ admits local transient and cyclicity.

▶ **Corollary 7.** An MPL system (2) is periodic iff all orbits $\pi \in \mathsf{Orb}(A)$ are lassos.

▶ Remark 8. The nature of the periodicity of an MPL system (2), as discussed above, plays an important role for the verification procedures in Section 4. For instance, over boundedly periodic MPL systems it entails the decidability of verification problems (cf. Corollary 18).

▶ **Example 9.** Consider a two-dimensional MPL system

$$\mathbf{x}(k+1) = A \otimes \mathbf{x}(k), \quad A = \begin{bmatrix} 2 & 5 \\ 3 & 3 \end{bmatrix}, \tag{3}$$

that models a simple railway network shown in Figure 2, where $x_i(k)$ represents the time of the k-th departure at station S_i for $i \in \{1, 2\}$. The element $A(i, j)$ for $i \neq j$ corresponds to the time needed to travel from station S_j to S_i. The element $A(i, i)$ represents the delay for the next departure of a train from station S_i.

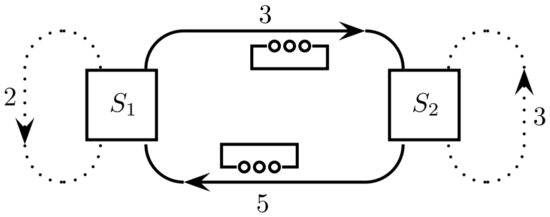


Figure 2 A simple railway network represented by an MPL system in (3).

Suppose the vector of initial departures is $\mathbf{x}(0) = [0\ 1]^\top$, then its corresponding orbit is

$$\pi = \begin{bmatrix}0\\1\end{bmatrix}, \begin{bmatrix}6\\4\end{bmatrix}, \begin{bmatrix}9\\9\end{bmatrix}, \begin{bmatrix}14\\12\end{bmatrix}, \dots$$

Notice that, the above orbit is periodic with transient $\mathsf{tr}(A, \mathbf{x}(0)) = 1$ and cyclicity $\mathsf{cyc}(A, \mathbf{x}(0)) = 2$, and is a $(2, 1)$-lasso.

2.2 Satisfiability Modulo Theory

Given a first-order formula ψ in a background theory T, Satisfiability Modulo Theory (SMT) refers to the problem of deciding whether there exists a model (i.e. an assignment to the free variables in ψ) that satisfies ψ [9]. For example, the formula $(x \le y) \wedge (x + 3 = z) \vee (z \ge y)$ within the theory of real numbers is satisfiable, and a valid model is $\{x := 5,\ y := 6,\ z := 8\}$.

SMT solvers can support different theories. A widely used theory is Quantifier-Free Linear Real Arithmetic (QF_LRA). A QF_LRA formula is an arbitrary Boolean combination of atoms in the form $\sum_i a_i \mathtt{x}_i \sim \alpha$, where $\sim \in \{>, \ge\}$, every $\mathtt{x}_i$ is a real variable, and every a_i and α are rational constants. Quantifier-Free Real Difference Logic (QF_RDL) is the subset of QF_LRA in which all atoms are restricted to the form $\mathtt{x}_i - \mathtt{x}_j \sim \alpha$. Both theories are decidable [9, Section 26.2]. Core to the main results of this work, it has been shown in [1] that any inequality in max-plus algebra can be translated into an RDL formula as follows.

▶ **Proposition 10** ([1]). Given real-valued variables $\mathtt{x}_1, \dots, \mathtt{x}_n$ and max-plus scalars $a_1, \dots, a_n$, $b_1, \dots, b_n \in \mathbb{R}_{\max}$, the inequality $F \equiv \bigoplus_{i=1}^{n} (\mathtt{x}_i \otimes a_i) \sim \bigoplus_{j=1}^{n} (\mathtt{x}_j \otimes b_j)$ is equivalent to $F^* \equiv \bigoplus_{i \in S_1} (\mathtt{x}_i \otimes a_i) \sim \bigoplus_{j \in S_2} (\mathtt{x}_j \otimes b_j)$, where $S_1 = \{1, \dots, n\} \setminus \{1 \le k \le n \mid a_k = \varepsilon \text{ or } \neg(a_k \sim b_k)\}$ and $S_2 = \{1, \dots, n\} \setminus \{1 \le k \le n \mid b_k = \varepsilon \text{ or } a_k \sim b_k\}$, respectively. Furthermore,

$$F^* \equiv \bigwedge_{j \in S_2} \left(\bigvee_{i \in S_1} (\mathtt{x}_i - \mathtt{x}_j \sim b_j - a_i) \right) \equiv \bigvee_{i \in S_1} \left(\bigwedge_{j \in S_2} (\mathtt{x}_i - \mathtt{x}_j \sim b_j - a_i) \right). \tag{4}$$

If $S_1 = \emptyset$ then $F^* \equiv \mathtt{false}$. On the other hand, if $S_2 = \emptyset$ then $F^* \equiv \mathtt{true}$.

Proposition 10 ensures that any inequality expression in max-plus algebra can be reduced to a simpler one in which no a variable appears on both sides. Then, the reduced inequality can be expressed as a QF_RDL formula either in conjunctive or disjunctive normal form[1].

[1] The readers are referred to the longer version of [1] in `https://arxiv.org/pdf/2007.00505.pdf` for a detailed proof.

As a direct consequence of Proposition 10, the MPL system dynamics in (2) can be expressed as a QF_RDL formula as follows:

$$\texttt{SymbMPL}(A, \mathcal{V}^{(k-1)}, \mathcal{V}^{(k)}) := \bigwedge_{i=1}^{n} (\texttt{ge}_i \wedge \texttt{eq}_i), \tag{5}$$

where $\texttt{ge}_i = \bigwedge_{j \in \texttt{fin}_i} (\mathtt{x}_i^{(k)} - \mathtt{x}_j^{(k-1)} \geq A(i,j))$ and $\texttt{eq}_i = \bigvee_{j \in \texttt{fin}_i} (\mathtt{x}_i^{(k)} - \mathtt{x}_j^{(k-1)} = A(i,j))$. The set $\mathcal{V}^{(k)} = \{\mathtt{x}_1^{(k)}, \ldots, \mathtt{x}_n^{(k)}\}$ contains SMT instances to encompass the states of the MPL system at the k-th bound while $\texttt{fin}_i$ is the indices of the finite elements of $A(i,\cdot)$.

3 Time-Difference Linear Temporal Logic

This section describes the logic we propose to express properties over MPL systems. We start by introducing the notions of Time-Difference (TD) proposition and of TD formula.

▶ **Definition 11.** A time-difference proposition over a vector of variables $\vec{\mathtt{x}} = \langle \mathtt{x}_1, \cdots, \mathtt{x}_n \rangle$ is an atomic formula $p = \mathtt{x}_i^{(k)} - \mathtt{x}_j^{(l)} \sim \alpha$, where $i, j \in \{1, \ldots, n\}, k, l \in \mathbb{N}, \alpha \in \mathbb{R}$ and $\sim \in \{>, \geq\}$. We call p *initial* if $k = l = 0$: for the sake of simplicity, we write $\mathtt{x}_i - \mathtt{x}_j \sim \alpha$ instead. For $m \geq 0$, we write $p^{(m)} = \mathtt{x}_i^{(k+m)} - \mathtt{x}_j^{(l+m)} \sim \alpha$.

▶ **Definition 12.** A TD formula for a vector of variables $\vec{x}$ is defined according to the following grammar

$$f ::= \texttt{true} \mid p \mid \neg f \mid f_1 \wedge f_2,$$

where $p = \mathtt{x}_i^{(k)} - \mathtt{x}_j^{(l)} \sim \alpha$ is a TD proposition over $\vec{\mathtt{x}}$. We call a TD formula f initial if all TD propositions appearing in f are initial ones.

Semantically, we interpret TD formulae on orbits[2]: given an orbit π and a TD formula f we say that $\pi \models f$ according to the following recursive rules.

$$\left.\begin{array}{ll} \pi \models \texttt{true} & \\ \pi \models \mathtt{x}_i^{(k)} - \mathtt{x}_j^{(l)} \sim \alpha & \text{iff } \pi[k]_i \sim \pi[l]_j + \alpha \\ \pi \models \neg f_1 & \text{iff } \pi \not\models f_1, \\ \pi \models f_1 \wedge f_2 & \text{iff } \pi \models f_1 \wedge \pi \models f_2. \end{array}\right\} \tag{6}$$

In the case of MPL systems, an orbit is uniquely determined by its initial vector $\mathbf{x}(0) = \pi[0]$; hence, one can write

$$\pi[0] \models f \text{ iff } \pi \models f, \tag{7}$$

or in general $\pi[i] \models f$ iff $\pi[i \ldots] \models f$. Finally, given an MPL system defined by matrix $A \in \mathbb{R}_{\max}^{n \times n}$ and a TD formula f, we say that $A \models f$ if all the orbits of A satisfy f (i.e., $\forall \pi \in \mathsf{Orb}(A). \pi \models f$).

Given an MPL system defined by matrix $A \in \mathbb{R}_{\max}^{n \times n}$ and a TD formula f, we can always rewrite f into an initial TD formula by means of the $\mathsf{get\_initial}(A, f)$ translation defined as follows. First, we translate all the TD propositions of f as the following inequality

$$\bigoplus_{r=1}^{n} \left(\mathtt{x}_r + A^{\otimes k}(i,r)\right) \sim \bigoplus_{s=1}^{n} \left(\mathtt{x}_s + \alpha + A^{\otimes l}(j,s)\right), \tag{8}$$

then, we can translate f into an initial TD formula by applying the rewriting (4) by Proposition 10.

[2] Equivalently, we could define the semantics on traces over $\vec{\mathtt{x}}$.

▶ **Proposition 13.** *Given a TD formula f and $A \in \mathbb{R}_{\max}^{n\times n}$, let g be* $\mathsf{get\_initial}(A, f)$. *Then, for any orbit $\pi \in \mathsf{Orb}(A)$, $\pi \models f$ iff $\pi \models g$.*

Therefore, each (non-initial) TD formula w.r.t. an MPL system characterised by $A \in \mathbb{R}_{\max}^{n\times n}$ can be translated into an initial one, with the same satisfaction relation over any orbit $\pi \in \mathsf{Orb}(A)$. It is important to note that the translation of a non-initial TD formula f into an initial TD formula g by Proposition 13 may result in a larger formula, in terms of the number of its propositions. Notice that the number of propositions in (4) is at most $\lfloor \frac{n}{2} \rfloor \times (n - \lfloor \frac{n}{2} \rfloor)$.

▶ **Proposition 14.** *Given a TD formula f and two similar orbits π, σ, $\pi \models f$ iff $\sigma \models f$.*

We are now in the position to discuss TD specifications, which generalise TD formulae to temporal requirements. Formally, TD specifications are defined as LTL formulae in Release Positive Normal Form (PNF) [8, Definition 5.23], according to the following grammar:

$$\varphi := \mathtt{true} \mid \mathtt{false} \mid p \mid \neg p \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \bigcirc \varphi \mid \varphi_1 \ \mathsf{U}\ \varphi_2 \mid \varphi_1 \ \mathsf{R}\ \varphi_2, \tag{9}$$

where p is an initial TD proposition. We call this logic Time-Difference Linear Temporal Logic (TDLTL). It is important to note that one can express a TDLTL specification that contains a non-initial TD formula f by first translating f into an initial TD formula, as per Proposition 13. Furthermore, any initial TD formula is a TDLTL formula (without any temporal operators). Given an orbit π, the semantics of TDLTL formulae (9) is defined as:

$$\left.\begin{array}{ll}
\pi \models \mathtt{true} & \text{for all } \pi \in \mathsf{Orb}(A),\\
\pi \not\models \mathtt{false} & \text{for all } \pi \in \mathsf{Orb}(A),\\
\pi \models p & \text{iff } \pi[0] \models p,\\
\pi \models \neg p & \text{iff } \pi \not\models p,\\
\pi \models \varphi_1 \wedge \varphi_2 & \text{iff } \pi \models \varphi_1 \wedge \pi \models \varphi_2,\\
\pi \models \varphi_1 \vee \varphi_2 & \text{iff } \pi \models \varphi_1 \vee \pi \models \varphi_2,\\
\pi \models \bigcirc\varphi & \text{iff } \pi[1..] \models \varphi,\\
\pi \models \varphi_1 \ \mathsf{U}\ \varphi_2 & \text{iff } \exists j \geq 0.\ \pi[j..] \models \varphi_2 \text{ and } \forall 0 \leq i < j.\ \pi[i\ldots] \models \varphi_1,\\
\pi \models \varphi_1 \ \mathsf{R}\ \varphi_2 & \text{iff } \forall j \geq 0.\ \pi[j] \models \varphi_2 \text{ or } \exists i \geq 0.\ (\pi[i..] \models \varphi_1 \wedge \forall h \leq i.\ \pi[h..] \models \varphi_2)\\
\pi \models \Diamond\varphi & \text{iff } \exists j \geq 0.\ \pi[j..] \models \varphi,\\
\pi \models \Box\varphi & \text{iff } \forall j \geq 0.\ \pi[j..] \models \varphi.
\end{array}\right\} \tag{10}$$

Similar to (7), the semantics of TDLTL formulae over initial vectors in the case of an MPL system is as follows: $\pi[0] \models \varphi$ iff $\pi \models \varphi$.

▶ **Example 15.** Consider a TDLTL formula $\varphi = \Diamond\Box(\bigwedge_{i=1}^{2}(3 \leq \mathrm{x}_i^{(1)} - \mathrm{x}_i^{(0)} \leq 5))$[3] defined for the MPL system in (3). The specification assesses whether the system eventually reaches a state after which the delays of consecutive departures from both stations are always between 3 and 5 time units. φ has four non-initial TD propositions; the "initialised" translations are:

$$\begin{array}{lclcl}
\mathrm{x}_1^{(1)} - \mathrm{x}_1^{(0)} \geq 3 & \Leftrightarrow & \max\{\mathrm{x}_1 + 2, \mathrm{x}_2 + 5\} \geq \mathrm{x}_1 + 3 & \Leftrightarrow & \mathrm{x}_2 - \mathrm{x}_1 \geq -2,\\
\mathrm{x}_2^{(1)} - \mathrm{x}_2^{(0)} \geq 3 & \Leftrightarrow & \max\{\mathrm{x}_1 + 3, \mathrm{x}_2 + 3\} \geq \mathrm{x}_2 + 3 & \Leftrightarrow & \mathtt{true},\\
\mathrm{x}_1^{(0)} - \mathrm{x}_1^{(1)} \geq -5 & \Leftrightarrow & \mathrm{x}_1 + 5 \geq \max\{\mathrm{x}_1 + 2, \mathrm{x}_2 + 5\} & \Leftrightarrow & \mathrm{x}_1 - \mathrm{x}_2 \geq 0,\\
\mathrm{x}_2^{(0)} - \mathrm{x}_2^{(1)} \geq -5 & \Leftrightarrow & \mathrm{x}_2 + 5 \geq \max\{\mathrm{x}_1 + 3, \mathrm{x}_2 + 3\} & \Leftrightarrow & \mathrm{x}_2 - \mathrm{x}_1 \geq -2.
\end{array}$$

Hence, the "initialised version" for φ is $\varphi' = \Diamond\Box((\mathrm{x}_2 - \mathrm{x}_1 \geq -2) \wedge (\mathrm{x}_1 - \mathrm{x}_2 \geq 0))$; equivalent to $\Diamond\Box(0 \leq \mathrm{x}_1 - \mathrm{x}_2 \leq 2)$.

[3] We write $k \leq x - y \leq w$ as a shorthand for $(x - y \leq w) \wedge (y - x \leq -k)$.

Given an MPL system (2), characterised by matrix $A \in \mathbb{R}_{\max}^{n\times n}$, and a TDLTL formula φ, we say that $A \models \phi$ if all the orbits of A satisfy ϕ (i.e., $\forall \pi \in \mathsf{Orb}(A).\ \pi \models \phi$). Furthermore, if the dynamics of (2) are defined over a set of initial conditions $X \subseteq \mathbb{R}^n$,

$$\mathsf{Orb}(A, X) \models \varphi \text{ iff } \forall \pi \in \mathsf{Orb}(A).\ \pi[0] \in X \implies \pi \models \varphi. \tag{11}$$

3.1 Encoding Bounded Counterexamples

This section describes the procedure to generate an SMT instance corresponding to a bounded counterexample of the TDLTL formula φ. By Corollary 7, such a counterexample can be generated over a lasso. First, we define the bounded version of (10) up to bound k for a (k, l)-lasso π as follows:

$$\left.\begin{array}{ll}
\pi \models_k p & \text{iff } \pi[0] \models p, \\
\pi \models_k \neg p & \text{iff } \pi \not\models_k p, \\
\pi \models_k \varphi_1 \wedge \varphi_2 & \text{iff } \pi \models_k \varphi_1 \wedge \pi \models_k \varphi_2, \\
\pi \models_k \varphi_1 \vee \varphi_2 & \text{iff } \pi \models_k \varphi_1 \vee \pi \models_k \varphi_2, \\
\pi \models_k \bigcirc\varphi & \text{iff } \pi[1..] \models_k \varphi, \\
\pi \models_k \varphi_1 \mathsf{U} \varphi_2 & \text{iff } \exists 0 \leq j \leq k.\ \pi[j..] \models_k \varphi_2 \text{ and } \forall 0 \leq i < j.\ \pi[i..] \models_k \varphi_1, \\
\pi \models_k \varphi_1 \mathsf{R} \varphi_2 & \text{iff } \forall 0 \leq j \leq k.\ \pi[j] \models_k \varphi_2 \text{ or} \\
 & \quad \exists 0 \leq i \leq k.\ (\pi[i..] \models_k \varphi_1 \wedge \forall h \leq i.\ \pi[h..] \models_k \varphi_2), \\
\pi \models_k \Diamond\varphi & \text{iff } \exists 0 \leq j \leq k.\ \pi[j,,] \models_k \varphi, \\
\pi \models_k \Box\varphi & \text{iff } \forall 0 \leq j \leq k.\ \pi[j..] \models_k \varphi.
\end{array}\right\} \tag{12}$$

Notice that, for a (k, l)-lasso π, $\pi[(k+1)..]$ is similar to $\pi[l..]$. Thus, it is straightforward to see that $\pi \models_k \varphi$ implies $\pi \models \varphi$.

▶ **Example 16.** For the TDLTL formula $\varphi' = \Diamond\Box(0 \leq \mathtt{x}_1 - \mathtt{x}_2 \leq 2)$ in Example 15 and a $(2, 1)$-lasso π in Example 9, one could check that $\pi \models_k \varphi'$ for $k = 2$.

We now describe how to translate a bounded counterexample of a TDLTL formula into an SMT instance. Suppose ψ is the negation of φ i.e. $\psi \equiv \neg\varphi$. We recall that φ and ψ are assumed to be in positive normal form (9). The notation ${}_l[\psi]_k^m$ denotes the witness encoding of ψ (equivalently, the counterexample encoding of φ) at position $0 \leq m \leq k$ over a (k, l)-lasso. Similar to the description in [10], the encoding can be formulated as follows:

$${}_l[p]_k^m := p^{(m)}, \qquad {}_l[\neg p]_k^m := \neg({}_l[p]_k^m),$$

$${}_l[\psi_1 \wedge \psi_2]_k^m := {}_l[\psi_1]_k^m \wedge\ {}_l[\psi_2]_k^m, \qquad {}_l[\psi_1 \vee \psi_2]_k^m :=\ {}_l[\psi_1]_k^m \vee\ {}_l[\psi_2]_k^m,$$

$${}_l[\bigcirc\psi]_k^m := \begin{cases} {}_l[\psi]_k^{m+1}, & \text{if } m < k \\ {}_l[\psi]_k^l, & \text{otherwise} \end{cases} \qquad {}_l[\Diamond\psi]_k^m := \bigvee_{j=\min\{m,l\}}^{k} {}_l[\psi]_k^j$$

$${}_l[\psi_1\ \mathsf{U}\ \psi_2]_k^m := \bigvee_{j=m}^{k} \left({}_l[\psi_2]_k^j \wedge \bigwedge_{n=m}^{j-1} {}_l[\psi_1]_k^n \right) \vee \qquad {}_l[\Box\psi]_k^m := \bigwedge_{j=\min\{m,l\}}^{k} {}_l[\psi]_k^j$$

$$\bigvee_{j=l}^{m-1} \left({}_l[\psi_2]_k^j \wedge \bigwedge_{n=m}^{k} {}_l[\psi_1]_k^n \wedge \bigwedge_{n=l}^{j-1} {}_l[\psi_1]_k^n \right)$$

$${}_l[\psi_1\ \mathsf{R}\ \psi_2]_k^m := \left(\bigwedge_{j=\min\{m,l\}}^{k} {}_l[\psi_2]_k^j \right) \vee \bigvee_{j=m}^{k} \left({}_l[\psi_1]_k^j \wedge \bigwedge_{n=m}^{j} {}_l[\psi_2]_k^n \right) \vee$$

$$\bigvee_{j=l}^{m-1} \left({}_l[\psi_1]_k^j \wedge \bigwedge_{n=m}^{k} {}_l[\psi_2]_k^n \wedge \bigwedge_{n=l}^{j} {}_l[\psi_2]_k^n \right),$$

where p is an initial TD proposition. The final formula, which is satisfiable iff there exists a (k, l)-lasso π such that $\pi \not\models_k \varphi$, is given by:

$$\bigwedge_{i=0}^{k} \mathtt{SymbMPL}(A, \mathcal{V}^{(i)}, \mathcal{V}^{(i+1)}) \wedge \mathsf{Loop}(A, k+1, l, \lambda) \wedge \ {}_l[\psi]_k^0, \tag{13}$$

where λ is the max-plus eigenvalue of A and $\mathsf{Loop}(A, k+1, l, \lambda)$ represents the looping constraint i.e., $\bigwedge_{i=1}^{n}(\mathtt{x}_i^{(k+1)} - \mathtt{x}_i^{(l)} = \lambda \times (k - l + 1))$. Recall that, if the orbit of $\mathbf{x}(0)$ w.r.t. A is (k, l)-lasso, then $\mathbf{x}(k+1) = (\lambda \times (k - l + 1)) \otimes \mathbf{x}(l)$. Furthermore, the first conjunct of (13) corresponds to the executions of (2) up to bound k.

By the same procedure used in Proposition 13, one can translate $\mathsf{Loop}(A, k+1, l, \lambda) \wedge {}_l[\psi]_k^0$ into an SMT formula over the variables $\mathcal{V}^{(0)}$ only (i.e., instead of representing the variables at each time in the orbit, we only define the formula over the initial state). Abusing the notation of TD formulae, we indicate the "initialised" version of (13) as

$$\mathsf{get\_initial}\left(A, \bigwedge_{i=0}^{k} \mathsf{Loop}(A, k+1, l, \lambda) \wedge \ {}_l[\psi]_k^0\right). \tag{14}$$

The first conjunct of (13) is not included because all TD propositions in (14) are initial ones. The number of TD propositions in (14) may be much larger compared to (13), especially for TDLTL formulae with multiple temporal operators. On the other hand, the encoding (14) has an advantage w.r.t. the number of variables: notice that (13) consists of variables from $\mathcal{V}^{(0)} \cup \ldots \cup \mathcal{V}^{(k+1)}$, whereas (14) involves $\mathcal{V}^{(0)}$ only.

3.2 An Upper Bound for the Completeness Threshold

The notion of *completeness threshold* refers to an index k such that, if no counterexample with length k or less for a LTL formula φ is found, then φ in fact holds over all infinite behaviours in the model. The discussion of how to compute the (upper bound of the) completeness threshold is given in [19, 31]. In this section, we show that the upper bound of the completeness threshold to verify (11) for any TDLTL formula over an MPL system (2) is determined by its pair transient/cyclicity.

▶ **Proposition 17.** Given a periodic MPL system (2) with a set of initial conditions X and a TDLTL formula φ, the upper bound of the completeness threshold to verify $\mathsf{Orb}(A, X) \models \varphi$ is given by $\mathsf{tr}(A, X) + \mathsf{cyc}(A, X) - 1$.

4 Model Checking of Max-Plus Linear Systems

This section describes the model-checking algorithms we devise for MPL systems. We build on the basic idea of BMC, that is to find a bounded counterexample of a given specification with a specific length k. If no such counterexample is found, then one increases k by one and searches corresponding counterexamples, until a known completeness threshold is reached, or until the problem becomes intractable. The reader is referred to [10, 11, 12] for a more detailed description.

Given an MPL system (2) with a set of initial conditions $X \subseteq \mathbb{R}^n$ and a TDLTL formula φ (9), we present procedures to verify whether $\mathsf{Orb}(A, X) \models \varphi$. In this paper, we assume that the underlying MPL system is periodic and the set of initial conditions X can be expressed as a general LRA formula. The procedures are integrated with the method computing the transient and cyclicity of MPL from a given set of initial conditions in [1]. We recall that such pair of transient and cyclicity can be used as an upper bound of the completeness threshold.

In the first part of Section 4, we present incremental algorithms to verify (11) where the bounded counterexample of a TDLTL formula φ is encoded for each iteration. Due to the periodic behaviour of MPL systems, such a counterexample corresponds to an orbit with transient l and cyclicity c. Unlike the usual BMC where the bound is increased by one, the novel procedures increase the step bound by finding the existence of a larger orbit with transient $l' > l$ or cyclicity c' not divides c. In the second part of the section, we describe "upfront" procedures where one only needs to encode the bounded counterexample of φ w.r.t. the upper bound of the completeness threshold given by Proposition 17.

4.1 Incremental Approaches

Orbits of MPL systems are potentially periodic with transient l and cyclicity c. Hence, in incremental procedures, we search a finite counterexample of a TDLTL formula φ with length k in a shape of (k,l)-lasso, where initially we set $l = 0$ and $c = 1$ (the smallest possible values). From these values, we generate the looping constraint $\mathsf{Loop}(A, k+1, l, \lambda)$ and ${}_l[\neg\varphi]_k^0$, where λ is the max-plus eigenvalue of A. The formula $X \wedge F$, with F given by (13) or (14), corresponds to a counterexample of φ i.e., a (k,l)-lasso $\pi \in \mathsf{Orb}(A, X)$, such that $\pi \not\models \varphi$.

We use an SMT solver to check the satisifiability of $X \wedge F$. If the SMT solver reports `SAT` then a counterexample is found. On the other hand, if the SMT solver reports `UNSAT`, we increment the step bound. Instead of increasing the bound by one, we use the SMT solver to check whether there exists an orbit with larger transient l' or cyclicity c'. The value of $l' + c' - 1$ becomes the new bound. These two steps are repeated until either 1) a counterexample is found; 2) the bound cannot be incremented; or 3) the bound is deemed too large. The second outcome suggests that the specification is valid, since the bound exceeds the upper bound of the completeness threshold given by Proposition 17. For the last outcome, we use a large integer as a termination condition.

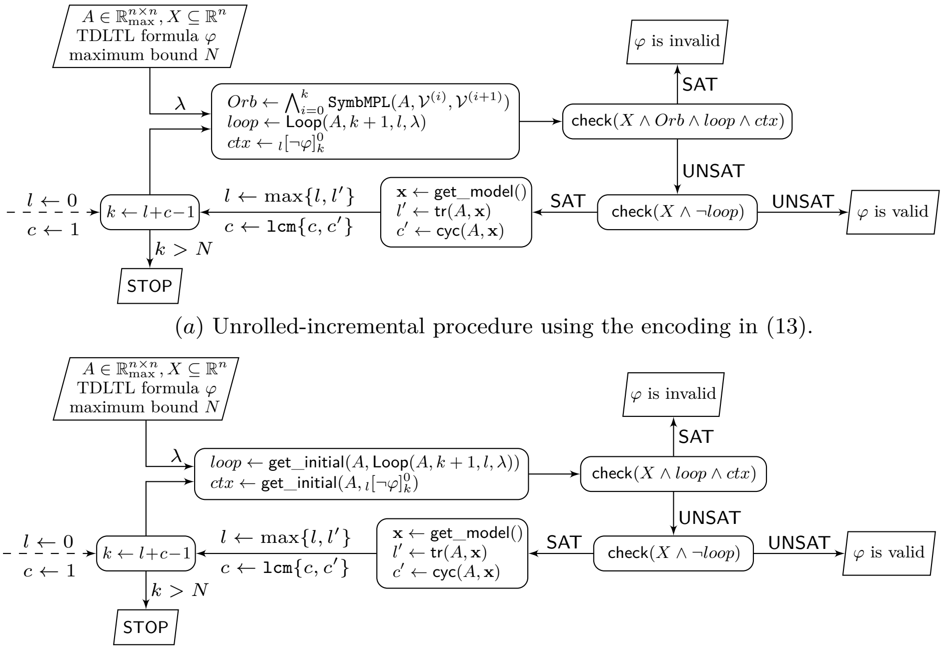
The incremental approaches are illustrated in Figure 3. We name the procedure that uses the encoding in (13) *unrolled-incremental*, since the first conjunct of (13) represents the execution of (2) up to bound k. On the other hand, the alternative procedure with the encoding in (14) is called *initialised-incremental*, due to the translation from Proposition 13.

4.2 Upfront Approaches

Upfront approaches exploit the fact that the upper bound of the completeness threshold in Proposition 17 is unrelated to the TDLTL formula φ, and that it can be computed via SMT-based techniques, as in [1, Algoritm 3]. Hence, the upfront versions of the procedures in Figure 3 is obtained by generating (13) or (14) with $l = \mathsf{tr}(A, X)$ and $k = \mathsf{tr}(A, X) + \mathsf{cyc}(A, X) - 1$. Together with the set of initial conditions X, we check the satisfaction of the resulting SMT instance. If it is satisfiable, then φ is invalid. On the other hand, if it is unsatisfiable, then φ is valid. The resulting procedures are then called *unrolled-upfront* and *initalised-upfront*, respectively.

4.3 Completeness and Decidability

Proposition 17 suggests that the completeness of the procedures in Sections 4.1-4.2 depends on the existence of $\mathsf{tr}(A, X)$ and $\mathsf{cyc}(A, X)$ for a given set X of initial conditions. From the discussed classification of MPL systems over their periodic behaviour, we can conclude that if (2) is boundedly periodic, then all procedures are complete. It is also shown in [1] that the computation of transient for unboundedly periodic MPL systems is semi-complete. We further summarise this discussion in Corollary 18.


$A \in \mathbb{R}_{\max}^{n\times n}, X \subseteq \mathbb{R}^n$
TDLTL formula φ
maximum bound N
λ
$Orb \leftarrow \bigwedge_{i=0}^{k} \texttt{SymbMPL}(A, \mathcal{V}^{(i)}, \mathcal{V}^{(i+1)})$
$loop \leftarrow \mathsf{Loop}(A, k+1, l, \lambda)$
$ctx \leftarrow {}_l[\neg\varphi]_k^0$
$\mathsf{check}(X \wedge Orb \wedge loop \wedge ctx)$
SAT
φ is invalid
UNSAT
$\mathsf{check}(X \wedge \neg loop)$
UNSAT
φ is valid
SAT
$\mathbf{x} \leftarrow \mathsf{get\_model}()$
$l' \leftarrow \mathsf{tr}(A, \mathbf{x})$
$c' \leftarrow \mathsf{cyc}(A, \mathbf{x})$
$l \leftarrow \max\{l, l'\}$
$c \leftarrow \texttt{lcm}\{c, c'\}$
$k \leftarrow l{+}c{-}1$
$l \leftarrow 0$
$c \leftarrow 1$
$k > N$
STOP

(*a*) Unrolled-incremental procedure using the encoding in (13).

$A \in \mathbb{R}_{\max}^{n\times n}, X \subseteq \mathbb{R}^n$
TDLTL formula φ
maximum bound N
λ
$loop \leftarrow \mathsf{get\_initial}(A, \mathsf{Loop}(A, k+1, l, \lambda))$
$ctx \leftarrow \mathsf{get\_initial}(A, {}_l[\neg\varphi]_k^0)$
$\mathsf{check}(X \wedge loop \wedge ctx)$
SAT
φ is invalid
UNSAT
$\mathsf{check}(X \wedge \neg loop)$
UNSAT
φ is valid
SAT
$\mathbf{x} \leftarrow \mathsf{get\_model}()$
$l' \leftarrow \mathsf{tr}(A, \mathbf{x})$
$c' \leftarrow \mathsf{cyc}(A, \mathbf{x})$
$l \leftarrow \max\{l, l'\}$
$c \leftarrow \texttt{lcm}\{c, c'\}$
$k \leftarrow l{+}c{-}1$
$l \leftarrow 0$
$c \leftarrow 1$
$k > N$
STOP

(*b*) Initialised-incremental procedure using the encoding in (14).

Figure 3 Incremental Approaches. The function check is implemented in an SMT solver. The integer N represents the allowed maximum bound.

▶ **Corollary 18.** Suppose we have an MPL system (2) with a set of initial condition X and a TDLTL formula φ. If the underlying MPL system is boundedly periodic (resp. unboundedly periodic) then the proposed procedures are complete (resp. semi-complete) and verifying $\mathsf{Orb}(A, X) \models \varphi$ is a decidable (resp. semi-decidable) problem.

5 Related Work

A verification by model checking procedure for MPL systems has been firstly discussed in [2]. It employs the *abstraction* [8, 27] of the underlying MPL system into an equivalent Piecewise Affine (PWA) model [28]. This results in an abstract model with a finite number of (abstract) states expressed as Difference Bound Matrices (DBM) [25, 33]. The approach allows to verify specifications (as LTL formulae) over the abstract model: if the specification is true, then it is also valid for the original MPL system [8]. However, the invalidity of specification on the abstract model does not necessarily imply the same conclusion on the concrete model. A refinement procedure is then proposed in [2]: the abstract model can be refined, so that it is in a bisimulation relation [8, Definition 7.1] with the original MPL system. This means that the specification is true on MPL system iff it holds on the bisimilar abstract model. Unfortunately, the proposed refinement procedure in general does not terminate, even for irreducible MPL systems. A sufficient condition for the existence of bisimilar abstract model is given in [2, Theorem 5]. The surveyed techniques suffer from the curse of dimensionality, since the abstraction computation runs on $\mathrm{O}(n^{(n+3)})$ time where n is the dimension of the state matrix.

The recent work [34] applies a different approach to verify MPL systems. A set of predicates is used to generate the abstraction of an MPL system. Predicates are automatically selected from the state matrix, as well as from the specifications under consideration. This *predicate abstraction* of MPL systems, reminiscent of [18, 27], is shown to be more scalable than the PWA-based abstraction in [2]. A standard BMC procedure is then applied to verify given specifications. It also has been shown in [34, Lemma 2] that the completeness threshold for this BMC procedure is determined by the pair of transient and cyclicity.

Despite successive ameliorations, the main drawback of the aforementioned methods is their scalability, as they can only be applied to MPL systems with relatively few variables (the dimension n of vector $\mathtt{x}$ in this work). There are a few elements contributing to the computational bottleneck (time and memory requirements) of these approaches. First, the worst-case complexity to generate the abstraction of n-dimensional MPL systems is $\mathrm{O}(n^{n+3})$ [2]. As a result, the number of abstract states grows exponentially, as n increases. Second, the refinement procedures in [2, 34] potentially lead to state-explosion problems. Another disadvantage is the limitations related to utilising the DBM data structure: in [34], each proposition in the form of $\mathtt{x}_i - \mathtt{x}_j \sim c$ where $1 \leq i, j \leq n, \sim \in \{>, \geq\}$ and $\alpha \in \mathbb{R}$ is transformed into a DBM in $\mathbb{R}^n$. As such, the more propositions are in an LTL formula, the more DBMs are needed, and therefore the larger number of abstract states.

6 Experiments

In our experimental evaluation, we compare the procedures introduced in Section 4 against alternative approaches based on the symbolic model checker nuXmv [15] and the existing abstraction-based techniques presented in [2, 33, 34].

6.1 Encoding in nuXmv (IC3)

We present a procedure to verify (11) using nuXmv [15], a symbolic model checker for the analysis of synchronous, finite- and infinite-state systems. This can be done by encoding the maximization operation into SMV language. For instance, $x'_1 = \max\{x_1 + a_1, x_2 + a_2\}$ can be expressed as:

```
TRANS ((next(x_1) >= (a_1 + x_1)) & (next(x_1) >= (a_2 + x_2))) &
      ((next(x_1) = (a_1 + x_1)) | (next(x_1) = (a_2 + x_2)));
```

Notice that the above expression is similar to (5). However, unlike the procedures in Figure 3, nuXmv requires that the lasso is in the canonical form. This means that, in Figure 1, the states $\mathbf{x}(l)$ and $\mathbf{x}(k+1)$ must be equal. This condition can be achieved if the corresponding matrix has eigenvalue equal to 0. It is straightforward that if matrix A in (2) has eigenvalue λ then $A \otimes (-\lambda)$ has eigenvalue 0.

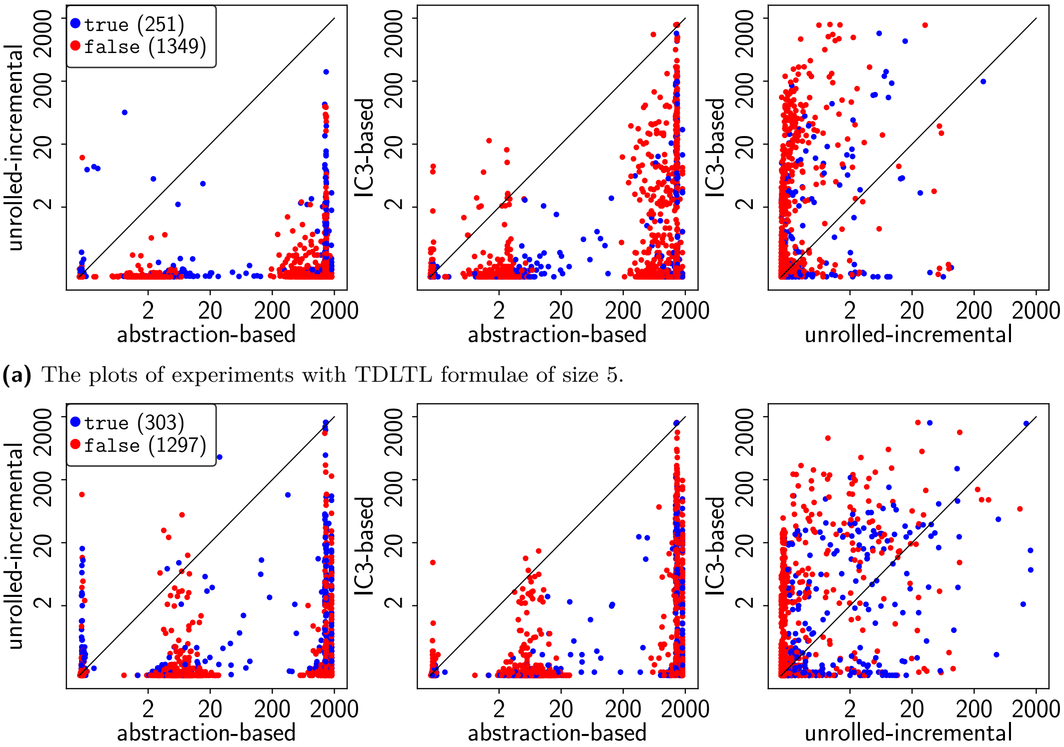
The procedure to verify (11) using nuXmv is as follows. First, we update the matrix by subtracting all elements with the corresponding eigenvalue. Then, we generate an SMV file from the matrix and the specification. Finally, we call nuXmv to verify the specification using command check_ltlspec_ic3 that implements the algorithm described in [22]. The algorithm works by trying to prove that any existing abstract fair loop is covered by a given set of well-founded relations, and leverages the efficiency and incrementality of the IC3ia [17] underlying safety checker.

6.2 Experimental Setup

For the experiments, we generate 20 irreducible matrices of dimension $n \in \{4, 6, 8, 10\} \cup \{12, 16, \ldots, 40\}$ with $\frac{n}{2}$ finite elements in each row, where the values of the finite elements are integers between 1 and 20. The locations of the finite elements are chosen randomly. We focus on irreducible matrices to ensure the termination of the procedures. With regards to the specifications, for each n, we generate randomly 20 TDLTL formulae where the propositions are in the form of $\mathtt{x}_i - \mathtt{x}_j \sim \alpha$, $i, j \in \{1, \ldots, n\}, \sim \in \{>, \geq\}$, and α is an integer within the interval $[-20, 20]$. The randomised TDLTL formulae are generated using Spot [26], which categorises them according to their size: namely, the size of a TDLTL formula φ is intended as number of operators and propositions in φ. For instance, the size of $p \wedge q$ and $p \mathsf{U} q$ is both 3 while $\Box p$ has size of 2. The experiments have been implemented using the SMT solver Z3 [23] on an Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz with 120GB of RAM memory. The set of initial conditions for each experiment is $X = \mathbb{R}^n$. The experiments are implemented for each pair of matrix and specification. Thus, there are $20 \times 20 \times 2$ experiments for each n. We set 30 minutes as a `timeout` limit.

6.3 Results

Figure 4 illustrates the performance comparison of abstraction-based, unrolled-incremental, and IC3-based algorithms for $n \in \{4, 6, 8, 10\}$. These three algorithms are similar in the sense that they unroll the dynamics of MPL systems up to the underlying step bound. The scattered plots (in logarithmic scale) represent the running times (in second) for a pair of algorithms. It is clear that the abstraction-based algorithm is in general the least efficient one. As expected, the dimension of MPL systems heavily affects the runtime of the abstraction-based procedure: all experiments for 10-dimensional MPL systems yield `timeout` (see Table 1 in Appendix B). For this reason, we do not pursue the abstraction-based experiments for higher dimensions.

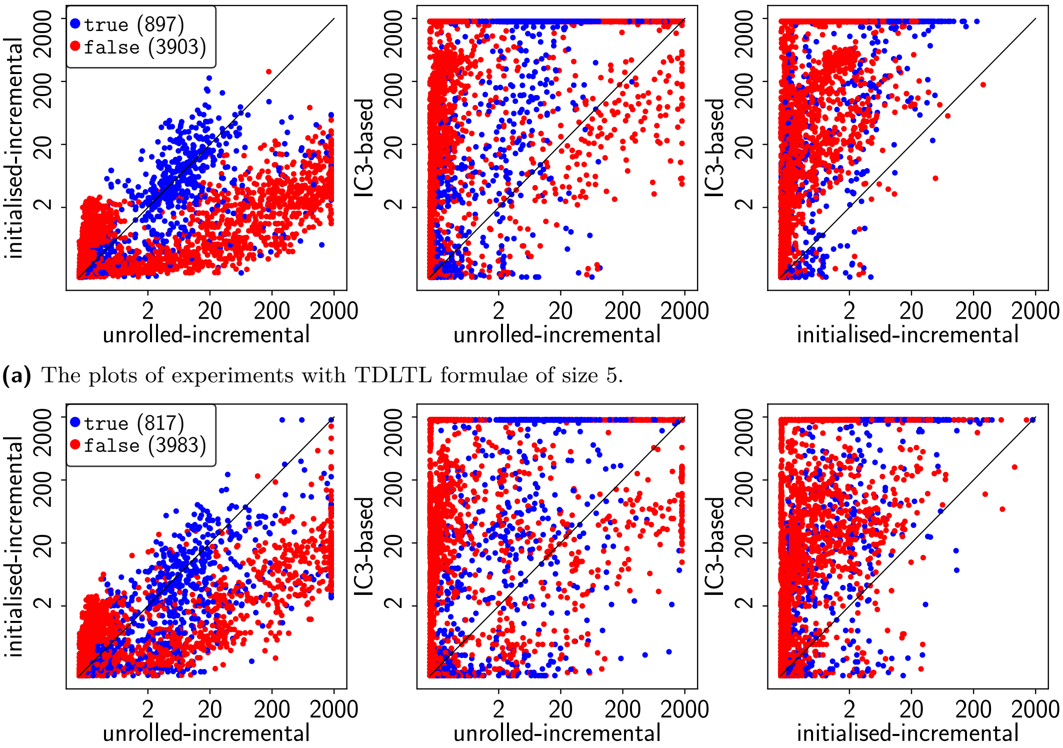


(a) The plots of experiments with TDLTL formulae of size 5.

(b) The plots of experiments with TDLTL formulae of size 10.

Figure 4 Comparison of abstraction-based, unrolled-incremental and IC3-based algorithms.

The plots in Figure 4 also indicate that the unrolled-incremental algorithm is more efficient than the IC3-based technique. To shed light on this finding, we then compare the performance of IC3-based technique with SMT-based incremental (unrolled and initialised) algorithms. As depicted in Figure 5, the proposed algorithms outperform the procedure that employs IC3. We recall that, in incremental algorithms, the bound of the counterexample is not increased by one. Hence, they are indeed more effective to find a long counterexample. Furthermore, in each iteration, the search of counterexample is implemented with a fixed loopback bound. Such a bound is related to the current value of transient l.



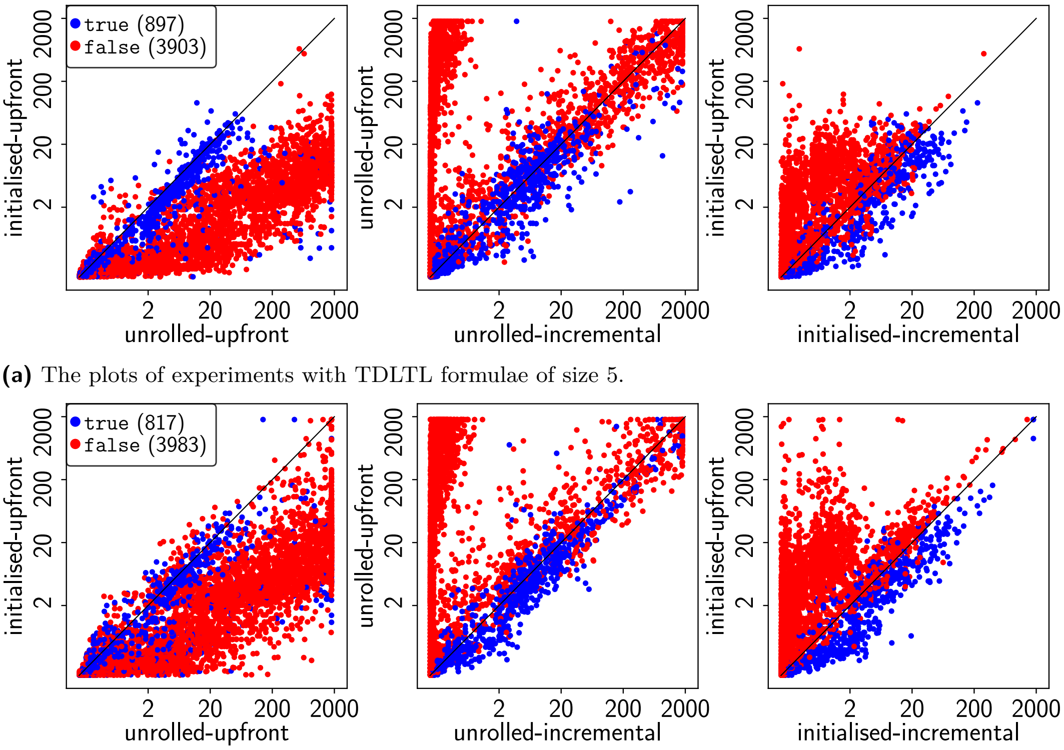
(a) The plots of experiments with TDLTL formulae of size 5.

(b) The plots of experiments with TDLTL formulae of size 10.

Figure 5 Comparison of incremental algorithms and IC3-based technique.

While the SMT-based incremental algorithms take a longer time to verify specifications with size of 10, the performance of the IC3-based algorithm is relatively similar. In fact, it is the dimension of the matrix which affects the running time of IC3-based procedure: the number of experiments that yield `timeout` increases as the dimension grows. On the other hand, the performance incremental algorithms are affected by the upper bound of completeness threshold given by Proposition 17; in particular for experiments whose corresponding specification is valid. Between the unrolled-incremental and initialised-incremental algorithms, it seems that the latter one is the winner. We recall that in Figure 3(b), all TD propositions in (14) are initial ones. Therefore, there are only one set of variables that appeared in (14). In comparison, there are $k+1$ sets of variables in (13) which correspond to the states of MPL system (2) from bound 0 until k.

We then compare the performance between incremental and upfront algorithms, as shown in Figure 6. As expected, upfront procedures are faster than incremental ones when the specification is valid. On the other hand, incremental algorithms are much more efficient when the specification is invalid. This is due to the fact that the counterexample may be found at a smaller bound than the completeness threshold given in Proposition 17. As in incremental approaches, the procedure which uses one set of variables (initialised-upfront) is faster than the other one that employs multiple sets of variables (unrolled-upfront).



(a) The plots of experiments with TDLTL formulae of size 5.

(b) The plots of experiments with TDLTL formulae of size 10.

Figure 6 Comparison of incremental and upfront algorithms.

7 Conclusions

In this paper, we addressed the problem of proving temporal properties over MPL systems. We defined TDLTL as a suitable formalism for the specification of temporal properties, and proposed a family of correct and complete SMT-based algorithms for the problem of checking TDLTL over MPL systems. We derived suitable completeness thresholds from the periodicity of MPL, and optimized the encoding by means of max-plus algebraic transformations. The results from a broad set of benchmarks demonstrate that the proposed approach can handle large models, which is completely out of reach for existing abstraction-based approaches, and that it outperforms a the baseline encoding into nuXmv.

As future work, we plan to investigate the case of reducible matrices, and to generalize the approach for *parametric* MPL, where the matrices may contain symbolic expressions.

References

1 Alessandro Abate, Alessandro Cimatti, Andrea Micheli, and Muhammad Syifa'ul Mufid. Computation of the transient in max-plus linear systems via SMT-solving. In Nathalie Bertrand and Nils Jansen, editors, *Formal Modeling and Analysis of Timed Systems*, pages 161–177, Cham, 2020. Springer International Publishing.

2 Dieky Adzkiya, Bart De Schutter, and Alessandro Abate. Finite abstractions of max-plus-linear systems. *IEEE Transactions on Automatic Control*, 58(12):3039–3053, 2013.

3 Dieky Adzkiya, Bart De Schutter, and Alessandro Abate. Backward reachability of autonomous max-plus-linear systems. *IFAC Proceedings Volumes*, 47(2):117–122, 2014.

4 Dieky Adzkiya, Bart De Schutter, and Alessandro Abate. Forward reachability computation for autonomous max-plus-linear systems. In Erika Abraham and Klaus Havelund, editors, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS'14), volume 8413 of *LNCS*, pages 248–262. Springer, 2014.

5 Dieky Adzkiya, Bart De Schutter, and Alessandro Abate. Computational techniques for reachability analysis of max-plus-linear systems. *Automatica*, 53:293–302, 2015.

6 Mohsen Alirezaei, Ton JJ van den Boom, and Robert Babuska. Max-plus algebra for optimal scheduling of multiple sheets in a printer. In *Proc. 31st American Control Conference (ACC), 2012*, pages 1973–1978, June 2012.

7 François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and linearity: an algebra for discrete event systems*. John Wiley & Sons Ltd, 1992.

8 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

9 Clark W. Barrett, Roberto Sebastiani, Sanjit A. Seshia, and Cesare Tinelli. Satisfiability modulo theories. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 825–885. IOS Press, 2009. doi:10.3233/978-1-58603-929-5-825.

10 Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 193–207. Springer, 1999. doi:10.1007/3-540-49059-0_14.

11 Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, Yunshan Zhu, et al. Bounded model checking. *Advances in Computers*, 58(11):117–148, 2003.

12 Armin Biere, Keijo Heljanko, Tommi Junttila, Timo Latvala, and Viktor Schuppan. Linear encodings of bounded LTL model checking. *Logical Methods in Computer Science*, 2(5):1–64, 2006. doi:10.2168/LMCS-2(5:5)2006.

13 J-L Bouquard, Christophe Lenté, and J-C Billaut. Application of an optimization problem in max-plus algebra to scheduling problems. *Discrete Applied Mathematics*, 154(15):2064–2079, 2006.

14 Chris A Brackley, David S Broomhead, M Carmen Romano, and Marco Thiel. A max-plus model of ribosome dynamics during mRNA translation. *Journal of Theoretical Biology*, 303:128–140, 2012.

15 Roberto Cavada, Alessandro Cimatti, Michele Dorigatti, Alberto Griggio, Alessandro Mariotti, Andrea Micheli, Sergio Mover, Marco Roveri, and Stefano Tonetta. The nuxmv symbolic model checker. In *International Conference on Computer Aided Verification*, pages 334–342. Springer, 2014.

16 Bernadette Charron-Bost, Matthias Függer, and Thomas Nowak. New transience bounds for max-plus linear systems. *Discrete Applied Mathematics*, 219:83–99, 2017.

17 Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. IC3 modulo theories via implicit predicate abstraction. In Erika Ábrahám and Klaus Havelund, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 46–61. Springer, 2014.

18 Edmund Clarke, Orna Grumberg, Muralidhar Talupur, and Dong Wang. Making predicate abstraction efficient. In *Proc. International Conference on Computer Aided Verification 2003 (CAV'03)*, pages 126–140. Springer, 2003.

19 Edmund Clarke, Daniel Kroening, Joël Ouaknine, and Ofer Strichman. Completeness and complexity of bounded model checking. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 85–96. Springer, 2004. doi:10.1007/978-3-540-24622-0_9.

20 J-P Comet. Application of max-plus algebra to biological sequence comparisons. *Theoretical computer science*, 293(1):189–217, 2003.

21 RA Cuninghame-Green and P Butkovic. Generalised eigenproblem in max-algebra. In *International Workshop on Discrete Event Systems*, pages 236–241. IEEE, 2008.

22 Jakub Daniel, Alessandro Cimatti, Alberto Griggio, Stefano Tonetta, and Sergio Mover. Infinite-state liveness-to-safety via implicit abstraction and well-founded relations. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, volume 9779 of *Lecture Notes in Computer Science*, pages 271–291. Springer, 2016.

23 Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Intl. Conf. on Tools and Algorithms for the Construction and Analysis of Systems* (TACAS'08), volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

24 Bart De Schutter. On the ultimate behavior of the sequence of consecutive powers of a matrix in the max-plus algebra. *Linear Algebra and its Applications*, 307(1-3):103–117, 2000.

25 David L Dill. Timing assumptions and verification of finite-state concurrent systems. In J. Sifakis, editor, *Intl. Conf. on Computer Aided Verification* (CAV'89), volume 407 of *Lecture Notes in Computer Science*, pages 197–212, Hiedelberg, 1989. Springer.

26 Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 - a framework for ltl and *omega*-automata manipulation. In *International Symposium on Automated Technology for Verification and Analysis*, pages 122–129. Springer, 2016.

27 Susanne Graf and Hassen Saïdi. Construction of abstract state graphs with PVS. In *In Proc. International Conference on Computer Aided Verification (CAV'97)*, pages 72–83. Springer, 1997.

28 Wilhemus Heemels, Bart De Schutter, and Alberto Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 37(7):1085–1091, July 2001.

29 Bernd Heidergott, Geert Jan Olsder, and Jacob Van der Woude. *Max Plus at work: modeling and analysis of synchronized systems: a course on Max-Plus algebra and its applications*. Princeton University Press, 2014.

30 Aleksey Imaev and Robert P Judd. Hierarchial modeling of manufacturing systems using max-plus algebra. In *Proc. American Control Conference, 2008*, pages 471–476, June 2008.

31 Daniel Kroening, Joël Ouaknine, Ofer Strichman, Thomas Wahl, and James Worrell. Linear completeness thresholds for bounded model checking. In *International Conference on Computer Aided Verification*, pages 557–572. Springer, 2011.

32 Glenn Merlet, Thomas Nowak, and Sergei Sergeev. Weak CSR expansions and transience bounds in max-plus algebra. *Linear Algebra and its Applications*, 461:163–199, 2014.

33 M.S. Mufid, D. Adzkiya, and A. Abate. Tropical abstractions of max-plus linear systems. In D.N. Jansen and P. Prabhakar, editors, *Int. Conf. Formal Modeling and Analysis of Timed Systems* (FORMATS'18), pages 271–287. Springer, 2018.

34 Muhammad Syifa'ul Mufid, Dieky Adzkiya, and Alessandro Abate. Bounded model checking of max-plus linear systems via predicate abstractions. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 142–159. Springer, 2019.

35 Thomas Nowak and Bernadette Charron-Bost. An overview of transience bounds in max-plus algebra. *Tropical and Idempotent Mathematics and Applications*, 616:277–289, 2014.

36 Gerardo Soto Y Koelemeijer. *On the behaviour of classes of min-max-plus systems*. PhD thesis, Delft University of Technology, 2003.

A Proofs of Propositions

▶ **Proposition 6.** An orbit π is a lasso iff $\pi[0]$ admits local transient and cyclicity.

Proof. ($\Leftarrow$) Suppose the transient and cyclicity for $\pi[0]$ is l anc c, respectively. Thus, $\pi[l+c+j] = \lambda c \otimes \pi[l+j]$ for $j \geq 0$ and π is a $(l+c-1, l)$-lasso.
($\Rightarrow$) Suppose π is a (k, l)-lasso. Then, there exists $\beta \in \mathbb{R}$ such that $\pi[k+1+j] = \beta \otimes \pi[l+j]$ for $j \geq 0$, or equivalently $\pi[j+k-l+1] = \beta \otimes \pi[j]$ for $j \geq l$. This means that $\mathsf{tr}(A, \pi[0]) = l$ and $\mathsf{cyc}(A, \pi[0]) = k-l+1$. ◀

▶ **Corollary 7.** An MPL system (2) is periodic iff all orbits $\pi \in \mathsf{Orb}(A)$ are lasso.

Proof. Direct consequence of Proposition 6. ◀

▶ **Proposition 13.** Given a TD formula f and $A \in \mathbb{R}_{\max}^{n \times n}$, let g be $\mathsf{get\_initial}(A, f)$. Then, for any $\pi \in \mathsf{Orb}(A)$, $\pi \models f$ iff $\pi \models g$.

Proof. Due to Definition 12 and (6), it suffices to prove for a non-initial TD proposition $p = \mathtt{x}_i^{(k)} - \mathtt{x}_j^{(l)} \sim \alpha$. Notice that that p is equivalent to an inequality (8) which can be translated into an initial TD formula by Proposition 10. This completes the proof. ◀

▶ **Proposition 14.** Given a TD formula f and two similar orbits π, σ, we have $\pi \models f$ iff $\sigma \models f$.

Proof. Suppose $\pi = \mathbf{x}(0)\mathbf{x}(1)\ldots$ and $\lambda = \mathbf{y}(0)\mathbf{y}(1)\ldots$ with $\mathbf{x}(m) = \beta \otimes \mathbf{y}(m)$ for $\beta \in \mathbb{R}$ and $m \geq 0$. The proof follows from the fact that for any TD proposition $p = \mathtt{x}_i^{(k)} - \mathtt{x}_j^{(l)} \sim \alpha$,

$$\begin{aligned}\pi \models p \text{ iff } & x_i(k) \sim x_j(l) + \alpha,\\ \text{iff } & y_i(k) + \beta \sim y_j(l) + \beta + \alpha,\\ \text{iff } & y_i(k) \sim y_j(l) + \alpha.\end{aligned}$$

The last assertion indicates that $\sigma \models p$. ◀

▶ **Proposition 17.** Given a periodic MPL system (2) with a set of initial conditions X and a TDLTL formula φ, the upper bound of completeness threshold to verify whether $\mathsf{Orb}(A, X) \models \varphi$ is given by $\mathsf{tr}(A, X) + \mathsf{cyc}(A, X) - 1$.

Proof. By Corollary 7, all orbits $\pi \in \mathsf{Orb}(A, X)$ are lasso. Recall that, if an initial vector $\mathbf{x}(0) \in X$ admits local transient $\mathsf{tr}(A, \mathbf{x}(0)) = l$ and cyclicity $\mathsf{cyc}(A, \mathbf{x}(0)) = c$ then the corresponding orbit is a (k, l)-lasso where $k = l + c - 1$. Consequently, the upper bound of completeness threshold to verify (11) is given by the largest possible of such k i.e., $\mathsf{tr}(A, X) + \mathsf{cyc}(A, X) - 1$. ◀

▶ **Corollary 18.** Suppose we have an MPL system (2) with a set of initial condition X and a TDLTL formula φ. If the underlying MPL system is boundedly periodic (resp. unboundedly periodic) then the proposed procedures are complete (resp. semi-complete) and verifying $\mathsf{Orb}(A, X) \models \varphi$ is a decidable (resp. semi-decidable) problem.

Proof. The completeness of the procedures follows from the fact that computing transient $\mathsf{tr}(A, X)$ is complete for boundedly periodic MPL systems, but semi-complete for unboundedly periodic ones. As a result, and because of Proposition 17, verifying $\mathsf{Orb}(A, X) \models \varphi$ is decidable for boundedly MPL systems and semi-decidable for unboundedly periodic ones. ◀

B The Runtime and Memory Consumption of Experiments

The following tables present the average and maximum running time (in second) and memory consumption (in MB) of the algorithms. The notation `timeout`(r) means that there are r (out of 400) failed experiments due to `timeout`.

Table 1 The runtime and memory consumption of abstraction-based algorithm.

n	Experiments with TDLTL formulae of size 5		Experiments with TDLTL formulae of size 10	
	runtime	memory	runtime	memory
4	$\{0.08, 1.42\}$	$\{26.29, 61.71\}$	$\{0.12, 12.11\}$	$\{28.3, 73.61\}$
6	`timeout`(9)	$\{69.45, 427.21\}$	`timeout`(5)	$\{110.84, 1238.51\}$
8	`timeout`(96)	$\{2715.27, 6401.11\}$	`timeout`(314)	$\{3517.04, 9837.71\}$
10	`timeout`(400)	$\{318.0, 569.61\}$	`timeout`(400)	$\{329.22, 608.91\}$

Table 2 The runtime and memory consumption of IC3-based algorithm.

n	Experiments with TDLTL formulae of size 5		Experiments with TDLTL formulae of size 10	
	runtime	memory	runtime	memory
4	$\{0.1, 8.89\}$	$\{9.94, 84.71\}$	$\{0.09, 9.69\}$	$\{11.21, 93.41\}$
6	$\{0.41, 22.71\}$	$\{22.34, 110.11\}$	$\{0.54, 24.81\}$	$\{21.32, 122.21\}$
8	$\{8.04, 1101.53\}$	$\{34.79, 275.71\}$	$\{5.02, 194.61\}$	$\{37.29, 170.11\}$
10	`timeout`(3)	$\{61.81, 262.21\}$	`timeout`(3)	$\{63.46, 1525.81\}$
12	`timeout`(12)	$\{69.12, 309.71\}$	`timeout`(10)	$\{71.51, 375.81\}$
16	`timeout`(150)	$\{96.37, 427.81\}$	`timeout`(115)	$\{92.51, 301.91\}$
20	`timeout`(243)	$\{105.9, 562.91\}$	`timeout`(233)	$\{105.39, 904.31\}$
24	`timeout`(189)	$\{100.78, 1326.61\}$	`timeout`(244)	$\{103.34, 345.71\}$
28	`timeout`(208)	$\{119.45, 1052.21\}$	`timeout`(255)	$\{104.97, 951.31\}$
32	`timeout`(228)	$\{125.82, 745.61\}$	`timeout`(254)	$\{103.79, 250.21\}$
36	`timeout`(232)	$\{126.33, 685.81\}$	`timeout`(293)	$\{118.79, 365.01\}$
40	`timeout`(245)	$\{149.08, 398.61\}$	`timeout`(268)	$\{154.99, 677.11\}$

Table 3 The runtime and memory consumption of unrolled-incremental algorithm.

n	Experiments with TDLTL formulae of size 5		Experiments with TDLTL formulae of size 10	
	runtime	memory	runtime	memory
4	$\{0.28, 63.77\}$	$\{6.63, 23.91\}$	$\{0.59, 116.33\}$	$\{9.24, 25.31\}$
6	$\{0.1, 5.62\}$	$\{8.84, 22.51\}$	$\{2.12, 457.88\}$	$\{9.23, 30.21\}$
8	$\{0.18, 3.37\}$	$\{10.99, 24.11\}$	$\{2.51, 227.76\}$	$\{11.62, 38.31\}$
10	$\{2.92, 280.91\}$	$\{22.94, 73.11\}$	`timeout`(1)	$\{23.78, 91.01\}$
12	$\{0.85, 34.69\}$	$\{13.76, 41.71\}$	$\{2.11, 120.39\}$	$\{13.44, 44.71\}$
16	$\{2.43, 57.04\}$	$\{21.3, 83.91\}$	$\{8.31, 1111.33\}$	$\{19.94, 78.61\}$
20	$\{13.79, 428.42\}$	$\{34.51, 230.91\}$	`timeout`(2)	$\{32.85, 220.91\}$
24	`timeout`(1)	$\{51.14, 544.51\}$	$\{29.31, 1130.4\}$	$\{42.59, 424.01\}$
28	`timeout`(17)	$\{120.89, 638.31\}$	`timeout`(14)	$\{66.89, 516.71\}$
32	`timeout`(16)	$\{132.17, 693.31\}$	`timeout`(24)	$\{77.32, 787.61\}$
36	`timeout`(31)	$\{123.9, 643.51\}$	`timeout`(19)	$\{82.76, 681.31\}$
40	`timeout`(71)	$\{179.88, 748.51\}$	`timeout`(73)	$\{152.97, 777.01\}$

Table 4 The runtime and memory consumption of initialised-incremental algorithm.

n	Experiments with TDLTL formulae of size 5		Experiments with TDLTL formulae of size 10	
	runtime	memory	runtime	memory
4	$\{0.03, 2.02\}$	$\{6.22, 22.61\}$	$\{0.15, 14.68\}$	$\{10.25, 23.41\}$
6	$\{0.07, 4.42\}$	$\{9.15, 22.61\}$	$\{0.46, 40.3\}$	$\{11.03, 23.41\}$
8	$\{0.1, 2.5\}$	$\{11.57, 22.51\}$	$\{0.39, 36.57\}$	$\{13.58, 23.41\}$
10	$\{0.35, 10.33\}$	$\{21.26, 23.41\}$	$\{5.18, 584.62\}$	$\{21.89, 23.91\}$
12	$\{0.39, 43.91\}$	$\{14.22, 22.21\}$	$\{0.64, 31.41\}$	$\{14.42, 23.41\}$
16	$\{0.32, 8.44\}$	$\{14.97, 18.81\}$	$\{1.2, 36.03\}$	$\{14.62, 18.81\}$
20	$\{0.82, 61.72\}$	$\{16.02, 61.71\}$	$\{1.47, 58.78\}$	$\{15.64, 23.21\}$
24	$\{1.65, 141.43\}$	$\{17.17, 181.51\}$	$\{1.49, 51.42\}$	$\{17.05, 29.41\}$
28	$\{3.27, 286.22\}$	$\{19.24, 30.41\}$	$\{3.8, 185.61\}$	$\{17.89, 58.01\}$
32	$\{2.56, 21.97\}$	$\{19.94, 29.71\}$	timeout(3)	$\{19.32, 58.61\}$
36	$\{8.1, 226.56\}$	$\{22.18, 41.31\}$	$\{8.0, 396.32\}$	$\{22.06, 52.31\}$
40	$\{8.93, 84.29\}$	$\{23.84, 43.01\}$	$\{18.64, 918.9\}$	$\{24.39, 50.91\}$

Table 5 The runtime and memory consumption of unrolled-upfront algorithm.

n	Experiments with TDLTL formulae of size 5		Experiments with TDLTL formulae of size 10	
	runtime	memory	runtime	memory
4	$\{0.41, 76.63\}$	$\{18.57, 27.21\}$	timeout(2)	$\{20.08, 34.01\}$
6	$\{0.44, 40.4\}$	$\{20.06, 29.11\}$	$\{12.6, 772.53\}$	$\{21.14, 33.41\}$
8	$\{0.63, 25.38\}$	$\{22.55, 34.31\}$	$\{8.98, 615.95\}$	$\{23.27, 39.11\}$
10	$\{4.54, 274.57\}$	$\{29.26, 81.31\}$	timeout(6)	$\{29.35, 98.21\}$
12	$\{1.44, 14.62\}$	$\{28.94, 53.81\}$	$\{7.41, 241.33\}$	$\{30.23, 55.41\}$
16	$\{7.15, 179.75\}$	$\{45.82, 169.31\}$	$\{17.22, 1122.95\}$	$\{44.93, 152.81\}$
20	$\{35.35, 610.82\}$	$\{87.02, 431.31\}$	timeout(4)	$\{81.37, 451.91\}$
24	timeout(1)	$\{142.51, 570.51\}$	timeout(2)	$\{138.93, 567.01\}$
28	timeout(3)	$\{223.51, 852.31\}$	timeout(14)	$\{241.62, 853.41\}$
32	timeout(30)	$\{307.83, 1095.81\}$	timeout(45)	$\{334.34, 1162.61\}$
36	timeout(29)	$\{381.8, 806.91\}$	timeout(40)	$\{395.85, 804.91\}$
40	timeout(125)	$\{587.1, 1096.21\}$	timeout(153)	$\{633.39, 1141.11\}$

Table 6 The runtime and memory consumption of initialised-upfront algorithm.

n	Experiments with TDLTL formulae of size 5		Experiments with TDLTL formulae of size 10	
	runtime	memory	runtime	memory
4	$\{0.05, 3.41\}$	$\{16.63, 22.81\}$	$\{4.49, 1393.45\}$	$\{18.43, 23.61\}$
6	$\{0.15, 10.63\}$	$\{18.75, 22.81\}$	$\{2.71, 730.72\}$	$\{18.92, 23.61\}$
8	$\{0.21, 3.94\}$	$\{18.97, 23.11\}$	$\{0.94, 44.02\}$	$\{20.22, 23.61\}$
10	$\{1.0, 183.35\}$	$\{21.68, 23.51\}$	timeout(1)	$\{22.24, 23.81\}$
12	$\{0.53, 15.0\}$	$\{20.64, 23.01\}$	$\{1.32, 20.2\}$	$\{21.15, 23.51\}$
16	$\{0.6, 9.53\}$	$\{21.01, 22.01\}$	$\{6.28, 1725.02\}$	$\{21.21, 22.81\}$
20	$\{1.66, 23.53\}$	$\{21.99, 24.51\}$	$\{5.05, 469.43\}$	$\{22.18, 24.51\}$
24	$\{3.3, 118.33\}$	$\{22.96, 27.71\}$	timeout(1)	$\{23.09, 28.01\}$
28	$\{7.15, 656.1\}$	$\{24.03, 32.61\}$	timeout(1)	$\{24.26, 34.41\}$
32	$\{4.77, 82.64\}$	$\{25.34, 35.81\}$	timeout(3)	$\{25.82, 36.61\}$
36	$\{12.3, 115.36\}$	$\{27.82, 42.21\}$	timeout(1)	$\{27.82, 41.61\}$
40	$\{15.82, 124.64\}$	$\{29.18, 47.41\}$	timeout(1)	$\{29.83, 47.21\}$

A Decidable Non-Regular Modal Fixpoint Logic

Florian Bruse ✉
School of Electrical Engineering and Computer Science, Universität Kassel, Germany

Martin Lange ✉
School of Electrical Engineering and Computer Science, Universität Kassel, Germany

Abstract

Fixpoint Logic with Chop (FLC) extends the modal μ-calculus with an operator for sequential composition between predicate transformers. This makes it an expressive modal fixpoint logic which is capable of formalising many non-regular program properties. Its satisfiability problem is highly undecidable. Here we define Visibly Pushdown Fixpoint Logic with Chop, a fragment in which fixpoint formulas are required to be of a certain form resembling visibly pushdown grammars. We give a sound and complete game-theoretic characterisation of FLC's satisfiability problem and show that the games corresponding to formulas from this fragment are stair-parity games and therefore effectively solvable, resulting in 2EXPTIME-completeness of this fragment. The lower bound is inherited from PDL over Recursive Programs, which is structurally similar but considerably weaker in expressive power.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Program specifications

Keywords and phrases formal specification, temporal logic, expressive power

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.23

1 Introduction

Modal fixpoint logics form the backbone of many program specification languages. The most prominent example is the modal μ-calculus $\mathcal{L}_\mu$ [15] which extends modal logic with least and greatest fixpoint quantifiers in order to express limit behaviour. Modal logic as a basis ensures bisimulation invariance which is a desirable property for program logics.

The two main decision problems associated with program logics are model checking and satisfiability checking. Whilst model checking for logics like $\mathcal{L}_\mu$ is easily seen to be decidable using fixpoint iteration for instance, satisfiability checking is computationally, combinatorically and conceptually more difficult. A relatively easy proof of the decidability of $\mathcal{L}_\mu$'s satisfiability problem is via a translation into Monadic Second-Order Logic (MSO) on trees using bisimulation-invariance [16]. MSO is decidable over infinite trees, but this only gives a non-elementary upper bound. Over time, this has been improved [26] until the problem could finally be placed in the complexity class EXPTIME using a translation into alternating tree automata [6].

The semantics of such automata can be phrased in terms of two-player games, and this can also be done directly to formulas. Hence, there are also game-theoretic characterisations of $\mathcal{L}_\mu$'s satisfiability problem as finite Rabin or parity games, which gives alternative decidability results [24, 9].

The EXPTIME upper bound is optimal; a matching lower bound is inherited from Propositional Dynamic Logic (PDL) – a modal logic that can be translated linearly into $\mathcal{L}_\mu$ and whose satisfiability problem is EXPTIME-complete [7]. The translation shows that PDL is in fact also a modal fixpoint logic with the fixpoints being implicitly present in the definition of regular languages used in modalities as in $\langle a^* b^* \rangle p$ for instance.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 23; pp. 23:1–23:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Regularity is also the answer to the question after $\mathcal{L}_\mu$'s expressiveness. The translation into MSO on trees shows that it can only express properties which are regular in the sense that they can be recognised by a finite automaton. $\mathcal{L}_\mu$ is in fact the largest such program logic as it is equi-expressive to the bisimulation-invariant fragment of MSO [12].

There is obviously a link between the decidability of satisfiability and the restriction of the expressiveness to regularity only. However, regularity on the expressiveness side does not constitute the border between decidability and undecidability, as there are program logics with the capability of expressing non-regular properties whose satisfiability problem remains decidable. The natural extension PDL[CFL] of PDL to *all* context-free (rather than just regular) programs is undecidable [10] but early on it has been observed that its extension by *some* context-free languages like $L_1 := \{a^n b^n \mid n \geq 1\}$ leads to a decidable program logic [14, 11].

Some such extension, for instance with the similar CFL $L_2 := \{a^n b a^n \mid n \geq 1\}$ lead to undecidability, though. This may seem odd at first sight but it makes perfect sense in the light of a later finding, namely that the extension of PDL with *all visibly pushdown* context-free languages PDL[VPL], is decidable [20]. Note that L_1 is a visibly pushdown language (VPL) but L_2 is not.

PDL[VPL] and $\mathcal{L}_\mu$ are incomparable in expressive power. The latter can express all regular and no non-regular properties, the former can express some non-regular *path* properties and by no means all regular properties. A typical PDL[VPL] property not expressible in $\mathcal{L}_\mu$ is $\langle L_1 \rangle p$ stating "*there is a path with a label from $a^n b^n$ ending in a p-state.*" A typical regular property not expressible in PDL[VPL] (or even its extension with the Δ-operator [25]) is $\mu X.\mathsf{win} \vee \Diamond\Box X$ defining the set of winning positions for the beginning player in a turn-based two-player game.

In this paper we show that the boundary of undecidability can be pushed up even further. We introduce a modal fixpoint logic which embeds both PDL[VPL] and $\mathcal{L}_\mu$ – the currently known peaks in the hierarchy of decidable program logics – and is thus strictly more expressive than either of them. The logic is obtained as a fragment of Fixpoint Logic with Chop (FLC) [23], an extension of $\mathcal{L}_\mu$ in which subformulas denote predicate transformers rather than predicates. Syntactically, this extension is comparable to the step taken from right-linear to context-free grammars, and the presence of conjunctions then makes satisfiability undecidable in general. However, we define a fragment by restricting the syntax such that, structurally, formulas resemble visibly pushdown grammars. This helps to retain decidability. We call this fragment Visibly Pushdown Fixpoint Logic with Chop (vpFLC). Conceptually, it is close to PDL[VPL] but the restriction to pure (visibly pushdown) path properties is lifted, and the typical visibly-pushdown principles can be combined more or less freely with modal operators to form properties like "*there is an $n \geq 1$ s.t. $\langle a \rangle^n [c] \langle b \rangle^n p$ holds.*" Henceforth, we will call this the $\langle a \rangle^n [c] \langle b \rangle^n$-property. From what seems to almost be a pure side-effect of this construction, vpFLC can express all regular properties, too. Hence, it genuinely pushes the limits of decidability amongst modal fixpoint logics as it extends both $\mathcal{L}_\mu$ and PDL[VPL] which are, as said above, incomparable in expressive power. Hence, vpFLC is at least as expressive as their union. Recently, it has even been shown to be strictly more expressive than this union, since there even are non-regular (and therefore non-$\mathcal{L}_\mu$-expressible) properties like $\langle a \rangle^n [b]^n p$ which which can expressed in vpFLC but not in PDL[CFL] and therefore also not in PDL[VPL] [1].

In Sect. 2 we recall necessary preliminaries from program logics, formal languages and games. In Sect. 3 we formally define vpFLC and argue that it extends both PDL[VPL] and $\mathcal{L}_\mu$ such that we obtain a 2EXPTIME lower bound for satisfiability inherited from the former. In Sect. 4 we define satisfiability games which are sound and complete for the full logic FLC, and show that for the fragment vpFLC they are decidable, leading to a matching 2EXPTIME upper bound. In Sect. 5 we conclude with remarks on further work in this area.

2 Preliminaries

2.1 Fixpoint Logic with Chop

Labelled transition systems. Let $\mathcal{P} = \{p, q, \ldots\}$ be a set of *atomic propositions* and $\Sigma = \{a, b, \ldots\}$ be a finite set of *action names.* A labeled transition system (LTS) over $\mathcal{P}$ and Σ is a $\mathcal{T} = (\mathcal{S}, \rightarrow, s_0, \lambda)$ s.t. $\mathcal{S}$ is a set of *states* with a designated initial state $s_0 \in S$, and $\rightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is the *transition relation.* We simply write $s \xrightarrow{a} t$ instead of $(s, a, t) \in \rightarrow$. Finally, $\lambda : \mathcal{S} \rightarrow 2^{\mathcal{P}}$ assigns a *label* to each state in the form of the atomic propositions which are supposed to be true in that respective state.

Syntax. Let $\mathcal{P}$ and Σ be as above. Let $\mathcal{V}$ be a countably infinite set of variable names. Formulas of FLC over $\mathcal{P}$, Σ and $\mathcal{V}$ are given by the following grammar.

$$\varphi \quad ::= \quad q \mid \overline{q} \mid X \mid \tau \mid \langle a \rangle \mid [a] \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mu X.\varphi \mid \nu X.\varphi \mid \varphi ; \varphi$$

where $q \in \mathcal{P}$, $a \in \Sigma$ and $X \in \mathcal{V}$.

We will write κ for either μ or ν. To save parentheses we introduce the convention that the operator ; binds stronger than $\wedge$ which binds stronger than $\vee$. We also use the abbreviations $\mathtt{ff} := q \wedge \overline{q}$ and $\mathtt{tt} := q \vee \overline{q}$ for some $q \in \mathcal{P}$. Note that FLC does not contain a negation operator to ensure well-definedness of fixpoints. Nevertheless, FLC is closed under complements, cf. [23] for details. Hence, we may also use Boolean operators like $\rightarrow, \leftrightarrow$ or even $\neg$ directly in examples when it helps to make formulas more understandable.

The set $\mathit{Sub}(\varphi)$ of subformulas of φ is defined as usual, with $\mathit{Sub}(\kappa X.\psi) = \{\kappa X.\psi\} \cup \mathit{Sub}(\psi)$ etc. For technical convenience with the satisfiability games introduced in Sect. 4, we assume that $\mathtt{tt}$ is always part of $\mathit{Sub}(\varphi)$ for any φ, cf. also the definition of the semantics below. Let $\mathit{Sub}^{\vee}(\varphi) = \{\psi_0 \vee \psi_1 \mid \psi_0 \vee \psi_1 \in \mathit{Sub}(\varphi)\}$ be the set of *disjunctive* subformulas of φ.

Formulas of the form q or $\overline{q}$ are called *literals*; those of the form $\langle a \rangle$ or $[a]$ are called *modalities.*

Formulas are assumed to be well-named in the sense that no variable is bound by a μ or a ν more than once in a given formula. Our main interest is with formulas that do not have free variables, in which case there is a function $\mathit{fp}_{\varphi} : \mathcal{V} \cap \mathit{Sub}(\varphi) \rightarrow \mathit{Sub}(\varphi)$ that maps each variable X to its unique defining fixpoint formula $\kappa X.\psi$ in φ.

Given two variables $X, Y \in \mathit{Sub}(\varphi)$ for some φ, we write $X <_{\varphi} Y$ if Y occurs free in $\mathit{fp}_{\varphi}(X)$. A variable X is called *outermost* among a set of variables $V \subseteq \mathcal{V} \cap \mathit{Sub}(\varphi)$ if it is maximal in V w.r.t. $<_{\varphi}$.

Semantics. Given an LTS $\mathcal{T} = (\mathcal{S}, \rightarrow, s_0, \lambda)$, an *environment* $\rho : \mathcal{V} \rightarrow (2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}})$ assigns to each variable a function from sets of states to sets of states in $\mathcal{T}$. We write $\rho[X \mapsto f]$ for the function that maps X to f and agrees with ρ on all other arguments.

The semantics $[\![\cdot]\!]^{\mathcal{T}}_{\rho} : 2^{\mathcal{S}} \rightarrow 2^{\mathcal{S}}$ of an FLC formula, relative to an LTS $\mathcal{T}$ and an environment, is such a function. It is monotone with respect to the inclusion ordering on $2^{\mathcal{S}}$. Such functions together with the partial order given by

$$f \sqsubseteq g \quad \text{iff} \quad \forall T \subseteq \mathcal{S} : f(T) \subseteq g(T)$$

form a complete lattice with joins $\sqcup$ and meets $\sqcap$ – defined as the pointwise intersection, resp. union. By the Knaster-Tarski Theorem [27] the least and greatest fixpoints of functionals $F : (2^{\mathcal{S}} \to 2^{\mathcal{S}}) \to (2^{\mathcal{S}} \to 2^{\mathcal{S}})$ exist. They are used to interpret fixpoint formulas of FLC. The semantics is then inductively defined as follows.

$$\begin{array}{rlrl}
\llbracket q \rrbracket^{\mathcal{T}}_{\rho} &= \_ \mapsto \{s \in \mathcal{S} \mid q \in \lambda(s)\} & \llbracket \langle a \rangle \rrbracket^{\mathcal{T}}_{\rho} &= T \mapsto \{s \in \mathcal{S} \mid \exists t \in T \text{ s.t. } s \xrightarrow{a} t\} \\
\llbracket \overline{q} \rrbracket^{\mathcal{T}}_{\rho} &= \_ \mapsto \{s \in \mathcal{S} \mid q \notin \lambda(s)\} & \llbracket [a] \rrbracket^{\mathcal{T}}_{\rho} &= T \mapsto \{s \in \mathcal{S} \mid \forall t : s \xrightarrow{a} t \Rightarrow t \in T\} \\
\llbracket Z \rrbracket^{\mathcal{T}}_{\rho} &= \rho(Z) & \llbracket \mu X.\varphi \rrbracket^{\mathcal{T}}_{\rho} &= \bigsqcap \{f : 2^{\mathcal{S}} \to 2^{\mathcal{S}} \mid f \text{ mon., } \llbracket \varphi \rrbracket^{\mathcal{T}}_{\rho[X \mapsto f]} \sqsubseteq f\} \\
\llbracket \tau \rrbracket^{\mathcal{T}}_{\rho} &= T \mapsto T & \llbracket \nu X.\varphi \rrbracket^{\mathcal{T}}_{\rho} &= \bigsqcup \{f : 2^{\mathcal{S}} \to 2^{\mathcal{S}} \mid f \text{ mon., } f \sqsubseteq \llbracket \varphi \rrbracket^{\mathcal{T}}_{\rho[X \mapsto f]}\} \\
\llbracket \varphi; \psi \rrbracket^{\mathcal{T}}_{\rho} &= \llbracket \varphi \rrbracket^{\mathcal{T}}_{\rho} \circ \llbracket \psi \rrbracket^{\mathcal{T}}_{\rho} & \llbracket \varphi \vee \psi \rrbracket^{\mathcal{T}}_{\rho} &= \llbracket \varphi \rrbracket^{\mathcal{T}}_{\rho} \sqcup \llbracket \psi \rrbracket^{\mathcal{T}}_{\rho} \\
 & & \llbracket \varphi \wedge \psi \rrbracket^{\mathcal{T}}_{\rho} &= \llbracket \varphi \rrbracket^{\mathcal{T}}_{\rho} \sqcap \llbracket \psi \rrbracket^{\mathcal{T}}_{\rho}
\end{array}$$

For any FLC formula φ, any LTS $\mathcal{T} = (\mathcal{S}, \to, s_0, \lambda)$ and any environment ρ let $\|\varphi\|^{\mathcal{T}}_{\rho} := \llbracket \varphi \rrbracket^{\mathcal{T}}_{\rho}(\mathcal{S})$. We call this the set of positions in $\mathcal{T}$ *defined* by φ and ρ. We also write $\mathcal{T}, s \models_{\rho} \varphi$ if $s \in \|\varphi\|^{\mathcal{T}}_{\rho}$, resp. $\mathcal{T} \models_{\rho} \varphi$ if $\mathcal{T}, s_0 \models_{\rho} \varphi$. If φ is closed we may omit ρ in both kinds of notation. We say that $\mathcal{T}$ is a *model* of a closed formula φ if $\mathcal{T} \models \varphi$. A formula is *satisfiable* if it has a model.

Two formulas φ and ψ are *equivalent*, written $\varphi \equiv \psi$, iff their semantics are the same, i.e. for every environment ρ and every LTS $\mathcal{T}$: $\llbracket \varphi \rrbracket^{\mathcal{T}}_{\rho} = \llbracket \psi \rrbracket^{\mathcal{T}}_{\rho}$. Two formulas φ and ψ are *weakly equivalent*, written $\varphi \approx \psi$, iff they define the same set of states in an LTS, i.e. for every ρ and every $\mathcal{T}$ we have $\|\varphi\|^{\mathcal{T}}_{\rho} = \|\psi\|^{\mathcal{T}}_{\rho}$. Hence, we have $\varphi \approx \varphi; \mathtt{tt}$ for any φ.

▶ **Example 1.** Take the $\langle a \rangle^n [c] \langle b \rangle^n$-property mentioned in the introduction. It is definable in FLC via $\varphi_{acb} := \langle a \rangle; (\mu X.[c] \vee \langle a \rangle; X; \langle b \rangle); \langle b \rangle; p$. For better readability, i.e. to stay closer to the syntax of more familiar fixpoint logics like $\mathcal{L}_{\mu}$ we drop the sequential composition operator when its left argument is a modality. This is semantically sound as the standard translation from $\mathcal{L}_{\mu}$ to FLC replaces $\langle a \rangle \varphi$ with $\langle a \rangle; \varphi$ etc. Then we can write this formula as $\langle a \rangle (\mu X.[c] \vee \langle a \rangle X; \langle b \rangle); \langle b \rangle p$.

To see that this truly formalises the desired property, we need two principles. They are simple consequences of the functional semantics and established truths in the theory of FLC.

1. Sequential composition is associative and left-commutes with Boolean operators, e.g. $(\varphi \vee \psi); \chi \equiv \varphi; \chi \vee \psi; \chi$.
2. The fixpoint unfolding principle holds, e.g. $\mu X.\varphi \equiv \varphi[\mu X.\varphi / X]$.

Then we have

$$\begin{aligned}
\langle a \rangle (\underbrace{\mu X.[c] \vee \langle a \rangle X; \langle b \rangle}_{\varphi_X}); \langle b \rangle p &\equiv \langle a \rangle ([c] \vee \langle a \rangle \varphi_X; \langle b \rangle); \langle b \rangle p \\
&\equiv \langle a \rangle [c] \langle b \rangle p \vee \langle a \rangle \langle a \rangle \varphi_X; \langle b \rangle \langle b \rangle p \\
&\equiv \langle a \rangle [c] \langle b \rangle p \vee \langle a \rangle \langle a \rangle ([c] \vee \langle a \rangle \varphi_X; \langle b \rangle); \langle b \rangle \langle b \rangle p \\
&\equiv \langle a \rangle [c] \langle b \rangle p \vee \langle a \rangle^2 [c] \langle b \rangle^2 p \vee \langle a \rangle^3 ([c] \vee \langle a \rangle \varphi_X; \langle b \rangle); \langle b \rangle^3 p \\
&\equiv \ldots \equiv \bigvee_{n \geq 1} \underbrace{\langle a \rangle \ldots \langle a \rangle}_{n} [c] \underbrace{\langle b \rangle \ldots \langle b \rangle}_{n} p
\end{aligned}$$

▶ **Example 2.** Consider $\varphi_{\mathsf{ubn}} := \nu X.\tau \wedge X; \langle a \rangle$. Remember that $\varphi_{\mathsf{ubn}} \approx \varphi_{\mathsf{ubn}}; \mathtt{tt}$ by definition of the semantics. Using a similar unfolding approach as above, we can see that

$$\varphi_{\mathsf{ubn}}; \mathtt{tt} \equiv \bigwedge_{n \geq 0} \langle a \rangle^n \mathtt{tt}$$

stating "*there are a-paths of unbounded lengths.*" This is not a regular property either: take for instance $\varphi_{\mathsf{ubn}}; \mathtt{tt} \wedge \mu Y.[a]Y$, i.e. its conjunction with the property stating that all a-paths are of finite length. This formula is satisfiable but does not have finite models, hence, $\varphi_{\mathsf{ubn}}; \mathtt{tt}$ cannot be expressed in $\mathcal{L}_\mu$ which has the finite model property.

Later, we will need the notion of guardedness in FLC formulas which plays the same role as it does in $\mathcal{L}_\mu$, requiring that fixpoint variables occur "behind" modal operators. For FLC, this is easy to define formally.

▶ **Definition 3.** *Let $\varphi \in$ FLC. An occurrence of a fixpoint variable X is* guarded*, if it occurs in the right argument ψ_2 of a sequential composition $\psi_1; \psi_2$ within its defining fixpoint formula $fp_\varphi(X)$ such that ψ_1 does not contain τ. The formula φ itself is guarded if all occurrences of fixpoint formulas in it are guarded.*

For instance, φ_{acb} from Ex. 1 is guarded but φ_{ubn} from Ex. 2 is not. This lifts the notion of guardedness – useful for decision procedures – from $\mathcal{L}_\mu$ to FLC in the most obvious way.

2.2 Visibly Pushdown Systems and Languages

A *visibly pushdown alphabet* is a partition $\Sigma = \Sigma_{\mathsf{int}} \uplus \Sigma_{\mathsf{call}} \uplus \Sigma_{\mathsf{ret}}$ of an action set into three categories of designated *internal*-, *call*-, resp. *return* symbols.

The concepts introduced in the following are always defined relative to a fixed visibly pushdown alphabet which will not be mentioned over and over again. Moreover, when connecting multiple ones, for instance when explaining equivalence between two visibly pushdown automata, the underlying alphabet is always assumed to be partitioned in the same way for all participating entities.

Visibly pushdown systems. A *visibly pushdown frame* (VPF) over a visibly pushdown alphabet Σ, partitioned accordingly, is a (Q, Γ, q_0, δ) where Q is a finite set of states, $q_0 \in Q$ is a designated starting state, Γ is a finite stack alphabet including a designated bottom-of-stack symbol $\bot$, and $\delta = \delta_{\mathsf{call}} \cup \delta_{\mathsf{int}} \cup \delta_{\mathsf{ret}}$ with

$$\delta_{\mathsf{call}} \subseteq Q \times \Gamma \times \Sigma_{\mathsf{call}} \times Q \times \Gamma \ , \quad \delta_{\mathsf{int}} \subseteq Q \times \Gamma \times \Sigma_{\mathsf{int}} \times Q \times \Gamma \ , \quad \delta_{\mathsf{ret}} \subseteq Q \times (\Gamma \setminus \{\bot\}) \times \Sigma_{\mathsf{ret}} \times Q$$

is its transition table. A *visibly pushdown system* (VPS) over a set $\mathcal{P}$ of atomic propositions is a $\mathcal{A} = (Q, \Gamma, q_0, \delta, \lambda)$ where (Q, Γ, q_0, δ) are a visibly-pushdown frame and $\lambda : Q \to 2^{\mathcal{P}}$ assigns to each state a set of atomic propositions.

Such a VPS gives rise to a generally infinite-state LTS $\mathcal{T}_{\mathcal{A}} = (Q \times \Gamma^*\bot, \to, (q_0, \bot), \lambda')$ where $\lambda'(q, \gamma) := \lambda(q)$, and transitions are given as follows. Let $q, p \in Q$, $B, C \in \Gamma$, $\gamma \in \Gamma^*$ and $a \in \Sigma$.

$$\begin{aligned} (q, B\gamma) &\xrightarrow{a} (p, C\gamma) && \text{if } (q, B, a, p, C) \in \delta_{\mathsf{int}} \\ (q, B\gamma) &\xrightarrow{a} (p, CB\gamma) && \text{if } (q, B, a, p, C) \in \delta_{\mathsf{call}} \\ (q, B\gamma) &\xrightarrow{a} (p, \gamma) && \text{if } (q, B, a, p) \in \delta_{\mathsf{ret}} \end{aligned}$$

Remember that $\Sigma_{\mathsf{call}} \cap \Sigma_{\mathsf{int}} = \emptyset$. Hence, if $(q, B, a, p, C) \in \delta_{\mathsf{call}}$ and $(q, B, a', p, C) \in \delta_{\mathsf{int}}$ then $a \neq a'$. The underlying directed graph $(Q \times \Gamma^*\bot, \to)$ is also called the *configuration graph* of the VPS $\mathcal{A}$.

A *path* in (the LTS associated with the) VPS $\mathcal{A}$ is, as usual, an infinite sequence alternating between states and actions $\pi = (q_0, \gamma_0), a_0, (q_1, \gamma_1), a_1, \ldots$ s.t. $(q_i, \gamma_i) \xrightarrow{a_i} (q_{i+1}, \gamma_{i+1})$ for all $i \geq 0$. We write $\mathit{Paths}(\mathcal{T}_{\mathcal{A}})$ for the set of all paths through $\mathcal{T}_{\mathcal{A}}$ starting in the initial state $(q_0, \bot)$.

For such a path π, the set of states (of the underlying VPS) occurring infinitely often is $\inf(\pi) = \{q \in Q \mid \text{there are infinitely many } i \text{ s.t. } q = q_i\}$. The *word* of such a path π is $\mathit{word}(\pi) = a_0 a_1 \ldots \in \Sigma^\omega$.

Let $(q_0, B_0\gamma_0), (q_1, B_1\gamma_1), \ldots$ be the projection of such a path π to the states, ignoring actions and giving names to the top stack symbols in each position. Such a position $(q_i, B_i\gamma_i)$ in π is a *stair-position* if for every $j \geq i$: $|\gamma_j| \geq |\gamma_i|$. I.e. in stair positions the top-most stack symbol may become replaced but not removed. Moreover the content of the stack below the top-most symbol persists throughout the entire remainder of the path since changing it would require the top-most stack symbol to be removed. For a path π, let $\mathit{stairs}(\pi)$ be the projection of π onto stair-positions, i.e. $\mathit{stairs}(\pi) = (q_0, \bot), (q_{i_1}, B_{i_1}\gamma_{i_1}), \ldots$ for some $0 < i_1 < i_2 < \ldots$. Note that each infinite path has infinitely many stair positions since the bottom-of-stack symbol $\bot$ never gets removed.

Visibly pushdown automata. A *visibly pushdown Büchi automaton* (VPA) is a $\mathcal{A} = (Q, \Gamma, q_0, \delta, F)$ s.t. (Q, Γ, q_0, δ) is a VPF and $F \subseteq Q$. Note that this can be seen as a VPS over a singleton atomic proposition fin marking those states that belong to F. Hence, a VPA also gives rise to an LTS $\mathcal{T}_\mathcal{A}$ just like a VPS does. The two are distinguished only because of their different pragmatic use: a VPS is a finite representation of an infinite-state LTS, serving as a model for branching-time properties expressible in logics like FLC; a VPA is a finite representation of an ω-language, obtained as follows.

The *language* of the VPA $\mathcal{A} = (Q, \Gamma, q_0, \delta, F)$ is

$$L(\mathcal{A}) = \{\mathit{word}(\pi) \mid \pi \in \mathit{Paths}(\mathcal{T}_\mathcal{A}), F \cap \inf(\pi) \neq \emptyset\}$$

consisting of all paths in the associated LTS which traverse states from F infinitely often.

A *stair-parity automaton* (SPA) is a $\mathcal{A} = (Q, \Gamma, q_0, \delta, \Omega)$ where (Q, Γ, q_0, δ) is a VPF and $\Omega : Q \to \mathbb{N}$. The *language* of an SPA is

$$L(\mathcal{A}) = \{a_0 a_1 \ldots \in \Sigma^\omega \mid \text{there is } \pi = (q_0, \gamma_0), a_0, (q_1, \gamma_1), a_1, \ldots \in \mathit{Paths}(\mathcal{T}_\mathcal{A}) \text{ s.t.}$$
$$\mathit{stairs}(\pi) = (q_{i_0}, \gamma_{i_0}), (q_{i_1}, \gamma_{i_1}), \ldots \text{ and } \limsup_{j \to \infty} \Omega(q_{i_j}) \text{ is even}\} .$$

Thus, a word $w = a_0 a_1 \ldots$ is accepted by a SPA if there is a path π through the LTS associated with the SPA s.t. the word along the transitions through π is w and the maximal priority which occurs infinitely often in stair positions along π is an even one.

An SPA $\mathcal{A} = (Q, \Gamma, q_0, \delta, \Omega)$ is *deterministic* (DSPA) if for every $q, p, p' \in Q$, $B, C, C' \in \Gamma$ and $a \in \Sigma$ we have

- if $(q, B, a, p, C), (q, B, a, p', C') \in \delta_{\mathsf{call}} \cup \delta_{\mathsf{int}}$ then $p = p'$ and $C = C'$, and
- if $(q, B, a, p), (q, B, a, p') \in \delta_{\mathsf{ret}}$ then $p = p'$.

The purpose of the introduction of the complicated stair-parity acceptance condition is the fact that (nondeterministic) VPA are not determinisable, not even with Muller acceptance conditions [2]. However, when the range of interpretation of the acceptance condition is restricted as it is done in SPA, then determinisation becomes possible. Moreover, DSPA are easy to complement.

We measure the size of a VPA or SPA over the VPF (Q, Γ, q_0, δ) as $|Q| + |\Gamma| + |\delta|$.

▶ **Proposition 4** ([21]). *For every VPA $\mathcal{A}$ of size n there is a DSPA $\mathcal{D}$ of size $2^{\mathcal{O}(n^2)}$ s.t. $L(\mathcal{D}) = \overline{L(\mathcal{A})}$.*

The main purpose of automata determinisation in the context of game solving is their ability to reduce games with abstract winning conditions to those with very concrete ones, see Cor. 6 below.

2.3 Games

Graph games. Two-player games played on directed graphs play a fundamental role in system's verification based on formal logic. We will use the game-theoretic approach to characterise FLC's satisfiability problem in Sect. 4 and to show decidability of the fragment defined in Sect. 3. This uses particular games known as stair-parity games defined below.

An (edge-labeled) *game* over a set Σ of labels is a $\mathcal{G} = (V, V_\exists, V_\forall, E, v_0, W)$ s.t. (V, E) is a directed graph. The nodes are also called *configurations* or *positions* of the game (arena). The edge relation $E \subseteq V \times \Sigma \times V$ is usually assumed to be total in the sense that for each $v \in V$ and $a \in \Sigma$ there is a $u \in V$ with $(v, a, u) \in E$. This is not a strict requirement, though, as there are easy ways to transform games without total edge relations into those with. $V_\exists$ and $V_\forall$ partition the set of positions into those owned by player $\exists$, resp. player $\forall$. $W \subseteq (V \times \Sigma)^\omega$ is the *winning condition* for player $\exists$.

A *play* is an alternating sequence of positions and actions $\pi = v_0, a_0, v_1, a_1, \ldots$ starting in the initial position v_0 and satisfying $(v_i, a_i, v_{i+1}) \in E$ for all $i \geq 0$. Player $\exists$ wins π if $\pi \in W$, otherwise it is won by player $\forall$.

A *strategy* for player $p \in \{\exists, \forall\}$ is a $\sigma_p : (V \times \Sigma)^*(V_p \times \Sigma) \to V$ s.t. for all $\rho = v_0, a_0, \ldots, v_n, a_n$ with $v_n \in V_p$ we have $(v_n, a_n, \sigma_p(\rho)) \in E$. I.e. it prescribes, given the *history* of a moment in a play, the next move to player p in terms of a successors of the current position and a given action. A play $\pi = v_0, a_0, v_1, a_1, \ldots$ *adheres* to σ_p if for every $n \in \mathbb{N}$ with $v_n \in V_p$ we have that $v_{n+1} = \sigma_p(v_0, a_0, \ldots, v_n, a_n)$, i.e. in this play, player p has always followed the advice given by σ_p whenever it was her turn to move.

The strategy σ_p is a *winning strategy* for player p, if every play that adheres to σ_p is winning for player p. The problem of *solving* is: given a game $\mathcal{G}$, decide whether or not player $\exists$ has a winning strategy for $\mathcal{G}$.

Stair-Parity Games. A *stair-parity game* (SPG) is a $\mathcal{G} = (Q, Q_\exists, Q_\forall, \Gamma, q_0, \delta, \Omega)$ such that $\mathcal{F} = (Q, \Gamma, q_0, \delta)$ is a VPF with its state space partitioned into $Q = Q_\exists \uplus Q_\forall$, and $\Omega : Q \to \mathbb{N}$ is a priority function as above. It gives rise to the abstract game $(V, V_\exists, V_\forall, v_0, E, W)$ where the arena (V, E) is the configuration graph of $\mathcal{F}$. The initial position is $v_0 = (q_0, \bot)$. The partition of positions belonging to either of the players is derived from the partition $Q_\exists \uplus Q_\forall$ into states belonging to them, s.t. $V_p = Q_p \times \Gamma^* \bot$ for $p \in \{\exists, \forall\}$.

The winning condition derived from Ω is defined similar to the acceptance condition for stair-parity automata, hence the name. A play $\pi = (q_0, \gamma_0), a_0, (q_1, \gamma_1), a_1, \ldots$ belongs to W iff $\limsup_{j \to \infty} \Omega(q_{i_j})$ is even where $i_0, i_1, i_2, \ldots$ is the set of stair positions in π.

The *size* of an SPG is simply defined as $|\delta|$. Note that an SPG can easily be represented using space that is linear in $|\delta|$. Another important parameter for the complexity of analysing games is its *index* which is defined as $|\{\Omega(q) \mid q \in Q\}|$.

▶ **Proposition 5** ([21]). *The problem of solving an SPG is decidable in* EXPTIME.

The proof is by a reduction to the problem of solving a parity game [22, 5] which is exponential in the size of the game but polynomial in its index. There are several algorithms showing that parity games can be solved in time polynomial in their size but exponential in their index, cf. [13], resulting in the EXPTIME upper bound for SPGs. Recent advances showing that parity games can be solved in quasi-polynomial [4] time do not have any impact that is significant for our purposes here.

▶ **Corollary 6.** *Suppose $\mathcal{G}$ is a game of size n, played on an arena that is a VPF with winning condition W s.t. $\overline{W}$ is recognised by a VPA of size $\mathcal{O}(n^c)$. Then $\mathcal{G}$ is solvable in* 2EXPTIME.

Proof. This uses the well-known product construction between a game and a deterministic automaton (cf. [9]) for the winning condition, obtained here through Prop. 4. If the latter is a DSPA, the resulting product becomes a stair-parity game of exponential size, and the doubly exponential bound follows from Prop. 5. ◀

3 Visibly Pushdown Fixpoint Logic with Chop

3.1 Syntax

The definition of the syntax of vpFLC is inspired by the structure of visibly pushdown grammars (VPG) [3] and PDL[VPL] (see Appendix A for a brief introduction). Again, we assume the set of actions to be partitioned into $\Sigma = \Sigma_{\mathsf{int}} \uplus \Sigma_{\mathsf{call}} \uplus \Sigma_{\mathsf{ret}}$.

▶ **Definition 7.** *The syntax of the fragment* vpFLC *of* FLC *is given by the following grammar.*

$$\begin{aligned}\varphi \quad ::= \quad & q \mid \overline{q} \mid X \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \mu X.\varphi \mid \nu X.\varphi \mid \\ & \{a_{\mathsf{int}}\} \mid \{a_{\mathsf{int}}\};\varphi \mid \{a_{\mathsf{call}}\};\{a_{\mathsf{ret}}\} \mid \{a_{\mathsf{call}}\};\varphi;\{a_{\mathsf{ret}}\} \mid \{a_{\mathsf{call}}\};\{a_{\mathsf{ret}}\};\varphi \mid \{a_{\mathsf{call}}\};\varphi;\{a_{\mathsf{ret}}\};\varphi\end{aligned}$$

where $q \in \mathcal{P}$, $X \in \mathcal{V}$, $a_x \in \Sigma_x$ for $m \in \{\mathsf{int}, \mathsf{call}, \mathsf{ret}\}$, and $\{a\}$ can be either $\langle a \rangle$ or $[a]$. Furthermore, we postulate that the sequential composition operator is right-associative; parentheses are not shown explicitly here for the sake of better readability.

Hence, vpFLC restricts the use of sequential composition such that the left argument is a modality over an internal or call-action. In the latter case, the right argument contains a modality over a return-action, possibly surrounded by formulas. The termination operator τ – the neutral element for sequential composition – is forbidden.

▶ **Example 8.** The $\langle a \rangle^n [c] \langle b \rangle^n$-property is definable in vpFLC provided that $a \in \Sigma_{\mathsf{call}}$, $b \in \Sigma_{\mathsf{ret}}$ and $c \in \Sigma_{\mathsf{int}}$. Reconsider the FLC formula φ_{acb} defined in Ex. 1. It is easily seen to be a vpFLC formula; note how the call-modality $\langle a \rangle$ is paired with the return-modality $\langle b \rangle$ both around the fixpoint variable X as well as the entire fixpoint formula for X.

The formula $\mu X.\langle b \rangle \vee \langle a \rangle; X; \langle a \rangle$ stating "*there is a path labeled $a^n b a^n$ for some $n \geq 0$*" is not a vpFLC formula because a cannot be a call- and return-modality at the same time, see also the comment on L_2 not being a VPL in the introduction.

3.2 Expressive Power

The logic vpFLC extends both $\mathcal{L}_\mu$ and PDL[VPL] in expressive power.

▶ **Theorem 9.** *For every formula $\varphi \in \mathcal{L}_\mu \cup \text{PDL[VPL]}$ there is a $\psi \in$ vpFLC s.t. $\psi \equiv \varphi$ and $|\psi| = \mathcal{O}(|\varphi|)$.*

Proof. (Sketch) The case of $\varphi \in \mathcal{L}_\mu$ is easy as $\mathcal{L}_\mu$ embeds into FLC straightforwardly via $\langle a \rangle \chi \equiv \langle a \rangle; \chi$ etc., cf. [23]. By considering all action symbols to be internal, $\Sigma = \Sigma_{\mathsf{int}}$, the resulting formulas fall into vpFLC. In fact, the standard translation produces formulas that fall into the syntax when restricted to literals, Boolean connectives, fixpoints and formulas of the form $\langle a \rangle; \varphi$ and $[a]; \varphi$ alone.

The other clauses in the syntax are needed to capture PDL[VPL]. We sketch the translation here for PDL[VPL] without the test operator. A corresponding extension is just a matter of technicality then. Take a PDL[VPL] formula of the form $\langle L \rangle \varphi$ where L is a VPL.

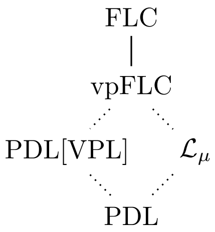


Figure 1 The expressiveness of logics closely related to vpFLC. Dotted lines are strict inclusions.

W.l.o.g. we can assume $\varepsilon \notin L$ because of $\langle\{\varepsilon\} \cup L\rangle\varphi \equiv \varphi \vee \langle L \setminus \{\varepsilon\}\rangle\varphi$. According to [3], L is representable by a context-free grammar with productions of the form $A \to \varepsilon$, $A \to a_{\mathsf{int}}B$, $A \to a_{\mathsf{call}}Ba_{\mathsf{ret}}C$. Using standard ε-elimination, this can be transformed into a grammar with productions in either of the following six forms.

$$A \to a_{\mathsf{int}}, \quad A \to a_{\mathsf{int}}B, \quad A \to a_{\mathsf{call}}a_{\mathsf{ret}}, \quad A \to a_{\mathsf{call}}Ba_{\mathsf{ret}}, \quad A \to a_{\mathsf{call}}a_{\mathsf{ret}}C, \quad A \to a_{\mathsf{call}}Ba_{\mathsf{ret}}C$$

Applying the translation of full PDL[CFL] into unrestricted FLC from [19] results in formulas which fall into vpFLC – note how its syntax resembles the form of these six grammar productions. ◀

For the embedding of PDL[VPL] it would indeed be easier to have τ in the syntax of vpFLC as this would eliminate the need to argue about the elimination of ε, and we would only need two instead of six patterns of productions. However, the inclusion of τ in vpFLC would invalidate the relatively simple argument in the proof of Lemma 19 below which is needed in the decidability proof.

Since PDL[VPL] and $\mathcal{L}_\mu$ are known to be incomparable in expressive power [20] we even have strict inclusion of both of them in vpFLC.

▶ **Corollary 10.** *Both* PDL[VPL] *and* $\mathcal{L}_\mu$ *are strictly less expressive than* vpFLC*.*

In fact, vpFLC is even more expressive than the union of PDL[VPL] and $\mathcal{L}_\mu$, cf. [1]. Fig. 1 shows the hierarchy of expressiveness amongst the modal fixpoint logics mentioned here. The undecidability of FLC– as opposed to vpFLC's decidability shown in the following section – gives a strong indication that FLC is strictly more expressive than vpFLC: there can, at least, be no *effective* translation from FLC to vpFLC.

Another consequence of the embeddings, which happen to be polynomial, is the inheritance of lower complexity bounds of the satisfiability problems from the embedded logics. For $\mathcal{L}_\mu$ it is "only" EXPTIME-hard, this is already the case for the smaller PDL [7]. However, the satisfiability problem for PDL[VPL] is even 2EXPTIME-hard [20].

▶ **Corollary 11.** *The satisfiability problem for* vpFLC *is* 2EXPTIME*-hard.*

4 Satisfiability Games

4.1 Satisfiability Games for FLC

Let $\chi \in$ FLC be closed. The *satisfiability game* $\mathcal{G}^{\mathsf{sat}}(\chi)$ is played between the *verifier* (**V**) and the *refuter* (**R**). A position in this game is a set of stacks, and a stack is a sequential composition $\psi_1; \ldots ; \psi_m$ of subformulas of χ. A position $\mathcal{C}$ is written $\gamma_1, \ldots, \gamma_n$, resp. $\varphi_1;\gamma_1$, ... , $\varphi_n;\gamma_n$ if we want to refer to the tops, resp. *heads* of the stacks particularly.

$$(\vee)\ \frac{(\varphi_0 \vee \varphi_1);\gamma\ ,\ \Phi}{\varphi_i;\gamma\ ,\ \Phi}\ \mathbf{V} : i \in \{0,1\} \qquad (\wedge)\ \frac{(\varphi_0 \wedge \varphi_1);\gamma\ ,\ \Phi}{\varphi_0;\gamma\ ,\ \varphi_1;\gamma\ ,\ \Phi} \qquad (;)\ \frac{(\varphi_0;\varphi_1);\gamma\ ,\ \Phi}{\varphi_0;\varphi_1;\gamma\ ,\ \Phi}$$

$$(\tau)\ \frac{\tau;\gamma\ ,\ \Phi}{\gamma\ ,\ \Phi} \qquad (\mathsf{fp})\ \frac{\kappa X.\varphi;\gamma\ ,\ \Phi}{X;\gamma\ ,\ \Phi} \qquad (\mathsf{var})\ \frac{X;\gamma\ ,\ \Phi}{\varphi;\gamma\ ,\ \Phi}\ \text{if } fp_\chi(X) = \kappa X.\varphi$$

$$(\mathsf{mod})\ \frac{\ell_1;\gamma_1,\dots,\ell_n;\gamma_n\ ,\ \langle a_1\rangle;\gamma'_1,\dots,\langle a_m\rangle;\gamma'_m\ ,\ [b_1];\gamma''_1,\dots,[b_k];\gamma''_k}{\gamma'_i, \{\gamma''_j \mid b_j = a_i\}}\ \mathbf{R} : i \in \{1,\dots,m\}$$

Figure 2 The rules of the FLC satisfiability games.

We may abbreviate several elements in a set of elements, writing such a position for instance as $\varphi;\gamma\ ,\ \Phi$, for instance when the particular shapes of elements other than $\varphi;\gamma$ are of no particular concern.

The *initial* position of the game $\mathcal{G}^{\mathsf{sat}}(\chi)$ is $\mathcal{C}_0 = \chi;\mathtt{tt}$, i.e. it only contains a single stack whose head is χ and rest is the single formula $\mathtt{tt}$. The game then proceeds according to the rules shown in Fig. 2. They all, apart from (mod), operate on the head of one stack, transforming this within a game position. If the head formula is a disjunction, then player $\mathbf{V}$ performs a choice with rule $(\vee)$ to replace it by one of its disjuncts. The others are deterministic and do not require a player to make a choice. Rule (mod), though, requires player $\mathbf{R}$ to make one. It is only applicable when all heads in the current position are either literals or modal formulas. The conclusion of this rule is called an a-child of its premise if $a_i = a$ for the i chosen by $\mathbf{R}$.

The next step to be taken is to explain under which condition a play is won by one of the players. This requires a technical definition, though.

▶ **Definition 12.** *Suppose $\pi = \Phi_0, \Phi_1, \dots$ is a* play *of $\mathcal{G}^{\mathsf{sat}}(\chi)$. A* thread *in π is a sequence $\pi = \gamma_0\ , \gamma_1, \dots$ of stacks s.t. for all $i \geq 0$: $\gamma_i \in \Phi_i$ and the rule played in Φ_i either*

- *operates on a formula different to the top of the stack γ_i and $\gamma_{i+1} = \gamma_i$, or*
- *operates on the top of γ_i, and γ_{i+1} results from γ_i through this rule application.*

Let $\varphi_0, \varphi_1, \dots$ be the sequence of topmost formulas in the stacks of π, i.e. $\gamma_i = \varphi_i;\gamma'_i$ for some γ'_i. Let $i_0, i_1, \dots$ be the set of stair positions in π' according to the definition in Sect. 2.2.

We say that π is a μ-thread, if the outermost variable occurring infinitely often in the sequence $\varphi_{i_0}, \varphi_{i_1}, \dots$ is of type μ. Otherwise, it is a ν-thread.

Threads trace the evolution of single formulas with their appended stacks through a play, and μ-threads witness the existence of a non-well-founded least fixpoint construct in the configurations of a play. These should be avoided in the search for a model. In other words, player $\mathbf{V}$– who aims to prove satisfiability of the input formula – should avoid μ-threads in her proof (in form of a winning strategy).

▶ **Definition 13.** *The play $\pi = \Phi_0, \Phi_1, \dots$ of $\mathcal{G}^{\mathsf{sat}}(\chi)$ is won by player $\mathbf{R}$, if*

- *there is an n s.t. $\Phi_n = q;\gamma\ ,\overline{q};\gamma'\ ,\Phi$ for some $q \in \mathcal{P}$ and some γ, γ', Φ; or*
- *π contains a μ-thread.*

Otherwise, player $\mathbf{V}$ wins π.

Next we show that the FLC satisfiability games are sound and complete. We make use of model checking games for FLC [17]; for convenience they are presented in Appendix B.

▶ **Theorem 14.** *If χ is satisfiable then player $\mathbf{V}$ has a winning strategy for $\mathcal{G}^{\mathsf{sat}}(\chi)$.*

Proof. Suppose there is an LTS $\mathcal{T}$ with a state s_0 s.t. $\mathcal{T}, s_0 \models \chi$. By definition we also have $\mathcal{T}, s \models \chi; \mathtt{tt}$. A strategy $\sigma_{\mathbf{V}}$ for $\mathbf{V}$ can be described as follows. She annotates each configuration $\gamma_1, \dots, \gamma_m$ with a state s from $\mathcal{T}$, written $s \vdash \gamma_1, \dots, \gamma_m$. We then call such an annotated configuration *safe* if $\mathbf{V}$ has a winning strategy in $\mathcal{G}^{\mathsf{mc}}_{\mathcal{T}}(\chi)$ starting from $s \vdash \gamma_i$ for all i. By completeness of the model checking games, she has a winning strategy σ' for $\mathcal{G}^{\mathsf{mc}}_{\mathcal{T}}(\chi)$ starting in $s_0 \vdash \chi; \mathtt{tt}$ and, hence, this annotated initial configuration is safe.

The key observation here is that $\mathbf{V}$ can preserve safety by consulting σ': whenever she is required to choose a disjunct in an annotated configuration $s \vdash (\varphi_0 \vee \varphi_1); \gamma, \Phi$, there is a corresponding choice of a disjunct under σ' in a configuration $s \vdash (\varphi_0 \vee \varphi_1); \gamma$. Moreover, the deterministic rules and $\mathbf{R}$'s choices preserve safety in general.

Now suppose $\pi = s_0 \vdash \Phi_0,\ s_1 \vdash \Phi_1, \dots$ is an (annotated) play adhering to this strategy $\sigma_{\mathbf{V}}$. It remains to be seen that it is won by $\mathbf{V}$. The second key observation is that any thread in this play, together with the state annotation corresponds to a play in $\mathcal{G}^{\mathsf{mc}}_{\mathcal{T}}(\chi)$ that adheres to σ'. By assumption, it is won by $\mathbf{V}$. If it is an infinite play then the outermost variable occurring infinitely often in stair positions is of type ν, i.e. it is a ν-thread. In other words, π cannot contain a μ-thread and is therefore won by $\mathbf{V}$. π cannot be won in finite time by player $\mathbf{R}$ either because there cannot be a safe configuration of the form $s \vdash q; \gamma,\ \overline{q}; \gamma,\ \Phi$, as by soundness of the model checking games we would have $s \models q$ and $s \not\models q$ at the same time. Hence, π is won by $\mathbf{V}$, showing that $\sigma_{\mathbf{V}}$ is indeed a winning strategy. ◀

Soundness does not hold in general: take for instance $(\nu X.X) \wedge (\mu Y.Y)$ which is unsatisfiable. Not every play will exhibit a μ-thread, though, as one may get caught up in the unwinding of $\nu X.X$. Guardedness (cf. Def. 3) excludes this as it requires rule (mod) to be played between each two unfoldings of any fixpoint formula.

▶ **Theorem 15.** *Let χ be guarded. If player $\mathbf{V}$ has a winning strategy for $\mathcal{G}^{\mathsf{sat}}(\chi)$ then χ is satisfiable.*

Proof. Consider the tree of all plays adhering to $\mathbf{V}$'s winning strategy σ. Guardedness ensures that each play uses rule (mod)infinitely often. Moreover, note that this tree only branches through applications of rule (mod) since this is the only one that gives player $\mathbf{R}$ the ability to perform choices.

We extract a tree model $\mathcal{T}_\sigma$ from this tree of plays by collapsing each segment $C_i, \dots, C_j$ into a state $\langle C_i, \dots, C_j \rangle$ s.t. before C_i and in C_j rule (mod) has been applied. Transitions are given as $\langle C_i, \dots, C_j \rangle \xrightarrow{a} \langle C_{j+1}, \dots, C_k \rangle$ if C_{j+1} is an a-child of C_j in rule (mod). The labelling is given by $q \in \lambda(\langle C_i, \dots, C_j \rangle)$ if $q; \gamma \in C_j$ for some γ. Write $\vec{C_0}$ for the initial state of $\mathcal{T}_\sigma$ according to this segmentation.

It remains to be seen that $\mathcal{T}_\sigma, \vec{C_0} \models \chi$. Suppose it was not the case. By soundness of $\mathcal{G}^{\mathsf{mc}}_{\mathcal{T}_\sigma}(\chi)$, $\mathbf{R}$ would have a winning strategy $\sigma_{\mathbf{R}}$ for this game starting in $\vec{C_0} \vdash \chi; \mathtt{tt}$. As in the proof of Thm. 14, the key observation is that any infinite play adhering to $\sigma_{\mathbf{R}}$ is a μ-thread in a play adhering to σ, contradicting the assumption that σ is a winning strategy for $\mathbf{V}$ in $\mathcal{G}^{\mathsf{sat}}(\chi)$. Moreover, suppose $\mathbf{R}$ won a finite play with $\sigma_{\mathbf{R}}$ by ending in a configuration $\vec{C} \vdash q; \gamma$, i.e. $\vec{C} \not\models q$ but $q \in \lambda(\vec{C})$ by construction. This also contradicts the assumption that σ would be a winning strategy. The other cases of winning finite plays are handled equally. Hence, we must indeed have $\mathcal{T}_\sigma, \vec{C_0} \models \chi$ finishing the proof. ◀

▶ **Example 16.** Consider $\varphi = (\mu Y.[c] \vee \nu X.\langle a \rangle; Y; \langle b \rangle; X) \wedge \nu Z.[a]; (Z \wedge \langle c \rangle; \mathtt{tt}); [b]$. The initial part of one play of the game $\mathcal{G}^{\mathsf{sat}}(\varphi)$ is shown in Fig. 3. For sake of brevity, rules that operate on separate stacks in a configuration have been compressed into a single step from one configuration to a next one, and the first rule application is not shown.

$$\begin{array}{rc}
 & (\mu Y.[c] \lor \nu X.\langle a\rangle;Y;\langle b\rangle;X);\mathtt{tt} \;,\; (\nu Z.[a];(Z \land \langle c\rangle;\mathtt{tt});[b]);\mathtt{tt} \\ \hline
\textsf{(fp)}, \textsf{(fp)} & Y;\mathtt{tt} \;,\; Z;\mathtt{tt} \\ \hline
\textsf{(var)}, \textsf{(var)} & ([c] \lor \nu X.\langle a\rangle;Y;\langle b\rangle;X);\mathtt{tt} \;,\; ([a];(Z \land \langle c\rangle;\mathtt{tt});[b]);\mathtt{tt} \\ \hline
(\lor), (;) & (\nu X.\langle a\rangle;Y;\langle b\rangle;X);\mathtt{tt} \;,\; [a];((Z \land \langle c\rangle;\mathtt{tt});[b]);\mathtt{tt} \\ \hline
\textsf{(fp)} & X;\mathtt{tt} \;,\; [a];((Z \land \langle c\rangle;\mathtt{tt});[b]);\mathtt{tt} \\ \hline
\textsf{(var)} & (\langle a\rangle;Y;\langle b\rangle;X);\mathtt{tt} \;,\; [a];((Z \land \langle c\rangle;\mathtt{tt});[b]);\mathtt{tt} \\ \hline
(;) & \langle a\rangle;(Y;\langle b\rangle;X);\mathtt{tt} \;,\; [a];((Z \land \langle c\rangle;\mathtt{tt});[b]);\mathtt{tt} \\ \hline
\textsf{(mod)} & (Y;\langle b\rangle;X);\mathtt{tt} \;,\; ((Z \land \langle c\rangle;\mathtt{tt});[b]);\mathtt{tt} \\ \hline
(;), (;) & Y;(\langle b\rangle;X);\mathtt{tt} \;,\; (Z \land \langle c\rangle;\mathtt{tt});[b];\mathtt{tt} \\ \hline
\textsf{(var)}, (\land) & ([c] \lor \nu X.\langle a\rangle;Y;\langle b\rangle;X);(\langle b\rangle;X);\mathtt{tt} \;,\; Z;[b];\mathtt{tt} \;,\; \langle c\rangle;\mathtt{tt};[b];\mathtt{tt} \\ \hline
(\lor), \textsf{(var)} & [c];(\langle b\rangle;X);\mathtt{tt} \;,\; ([a];(Z \land \langle c\rangle;\mathtt{tt});[b]);[b];\mathtt{tt} \;,\; \langle c\rangle;\mathtt{tt};[b];\mathtt{tt} \\ \hline
(;) & [c];(\langle b\rangle;X);\mathtt{tt} \;,\; [a];((Z \land \langle c\rangle;\mathtt{tt});[b]);[b];\mathtt{tt} \;,\; \langle c\rangle;\mathtt{tt};[b];\mathtt{tt} \\ \hline
\textsf{(mod)} & (\langle b\rangle;X);\mathtt{tt} \;,\; \mathtt{tt};[b];\mathtt{tt} \\ \hline
(;) & \langle b\rangle;X;\mathtt{tt} \;,\; \mathtt{tt};[b];\mathtt{tt} \\ \hline
\textsf{(mod)} & X;\mathtt{tt} \\ \hline
 & \vdots
\end{array}$$

Figure 3 Initial part of a play on the formula from Ex. 16 with some rule applications compressed.

It would be cumbersome to try to express the property formalised by φ in English words; in fact, φ is constructed specifically to exemplify the mechanics of the satisfiability games. The grey background highlights the evolution of one of the stacks, particularly its elements up to stack height 2. One can see that the stack reaches height 3 inbetween, and it is there that Y occurs in head position for the second time. This is not a stair position, though. The play could be continued s.t. both X and Y occur infinitely often in head positions but it is only the inner X which does so in stair positions. Hence, this forms a ν-thread.

4.2 Satisfiability Games for Guarded vpFLC are Stair-Parity Games

Thms. 14 and 15 give a reduction from FLC's satisfiability problem to the problem of solving particular games on infinite-state spaces. Undecidability of the former transfers to these games which are therefore not algorithmically solvable for arbitrary FLC formulas. They are, however, for guarded vpFLC formulas. We prove this by revealing them as special stair-parity games. As a first step in this direction, we reformulate them as turn-based games. Fix some $\chi \in$ vpFLC over some visibly pushdown alphabet Σ.

▶ **Definition 17.** *Configurations of the* turn-based satisfiability game $\mathcal{G}^{\mathsf{sat}}_{\mathsf{tb}}(\chi)$ *are of the same form as those in the original* FLC *satisfiability game* $\mathcal{G}^{\mathsf{sat}}(\chi)$*. The initial configuration is also* $\chi;\mathtt{tt}$*. The rules, however, deviate.*

Let $f : Sub^{\lor}(\chi) \to \{0,1\}$*. A configuration* C' *is the* f*-normalisation of a configuration* C*, if* C' *is obtained by repeatedly applying rules* $(\land)$, $(;)$, $\textsf{(fp)}$ *and* $\textsf{(var)}$ *to* C *until all heads of all stacks are either literals or modalities. Likewise, when some head of a stack is of a form* $\psi_0 \lor \psi_1$ *then it gets replaced by* ψ_i *using rule* $(\lor)$ *where* $i = f(\psi_0 \lor \psi_1)$*.*

In a configuration C*, the turn-based game* $\mathcal{G}^{\mathsf{sat}}_{\mathsf{tb}}(\chi)$ *proceeds as follows.*

1. $\mathbf{V}$ *selects a function* $f : Sub^{\lor}(\chi) \to \{0,1\}$*,*
2. $\mathbf{R}$ *applies rule* $\textsf{(mod)}$ *to the* f*-normalisation of* C*.*

The following is not difficult to see; $\mathcal{G}^{\mathsf{sat}}_{\mathsf{tb}}(\chi)$ can be seen as an efficient way of playing $\mathcal{G}^{\mathsf{sat}}(\chi)$ with several rules being played at once.

▶ **Lemma 18.** *Player* $\mathbf{V}$ *wins* $\mathcal{G}^{\mathsf{sat}}_{\mathsf{tb}}(\chi)$ *iff she wins* $\mathcal{G}^{\mathsf{sat}}(\chi)$.

Proof. Note that all that the definition of $\mathcal{G}^{\mathsf{sat}}_{\mathsf{tb}}(\chi)$ does is to condense consecutive choices $\mathbf{V}$ does in selecting disjuncts and the applications of deterministic rules into a single move by player $\mathbf{V}$. All that is needed then is to see that the syntax of vpFLC guarantees the f-normalisation of any configuration to exist (uniquely), for any f. In a sense, vpFLC formulas are guarded, i.e. between two unfoldings of a fixpoint variable, rule (mod) needs to be played. In $\mathcal{G}^{\mathsf{sat}}_{\mathsf{tb}}(\chi)$, this just happens in two consecutive steps requiring a $\mathbf{V}$ choice followed by a $\mathbf{R}$ choice. Hence, winning strategies can easily be transferred between these two games. ◀

The next goal is to show that turn-based satisfiability games are in fact stair-parity games. There are two obstacles to overcome. Note that positions in stair-parity games contain a single stack, whereas satisfiability games on arbitrary FLC formulas can contain an unbounded number of stacks.

1. There needs to be a representation of the set of stacks as a single one. This is possible because on vpFLC formulas, the sizes of all stacks do not deviate by much. Secondly, we can use a standard encoding via transition profiles in order to represent the unboundedly many stacks by one of fixed width.
2. The winning condition for $\mathbf{V}$ needs to be phrased as a stair-parity condition.

To tackle the first problem, we call a configuration $\gamma_1, \ldots, \gamma_m$ *k-aligned* for $k \geq 0$, if the stack heights differ by at most k: $||\gamma_i| - |\gamma_j|| \leq k$ for all i, j. Note that the initial configuration of $\mathcal{G}^{\mathsf{sat}}_{\mathsf{tb}}(\chi)$ is always 0-aligned as it contains only one stack. The first key observation here is the following. It uses the fact that sequential composition is forced by the syntax to be right-associative, and that a child under the modal rule is built from stacks all starting with modalities for the same action symbol.

▶ **Lemma 19.** *The following holds.*

a) *The f-normalisation C' of a 0-aligned configuration C is 1-aligned, and*

b) *the a-child C'' of any 1-aligned configuration C' in rule* (mod) *is 0-aligned, for any $a \in \Sigma$.*

Proof. Part (b) is easier to see: note that all stacks in C'' are suffixes of a stack in C' which started with a modality containing a. The syntax of vpFLC can only create non-0-aligned configurations by making use of two different actions, though. Note how, for instance, a call-modality $\langle a \rangle$ can only be used in a formula of the form $\alpha_{\mathsf{call}}; \varphi; \alpha_{\mathsf{ret}}$ according to the syntax, and this is the case for any other stack head which is either $\langle a \rangle$ or $[a]$ for the same $a \in \Sigma$.

For part (a) observe that sequential compositions in the syntax of vpFLC (apart from those directly linked to call- and return-modalities) are right-associative. Hence, rule (;) can increase the size of a stack by one but not any more as further sequential compositions can only occur in the right argument but not the left. ◀

This reduces the complexity of a potentially unbounded number of stacks of different height to a potentially unbounded number of stacks of essentially the same height. Next we observe that it can be reduced further to a bounded number of stacks of (almost) equal height. This follows from the fact that the game rules operate on the stack heads only of which there are at most n different ones. Additional stacks can only be created using rule ($\wedge$) which duplicates the rest of the stack that it operates on.

▶ **Lemma 20.** *Let $C = \gamma_1, \dots, \gamma_m$ be a configuration of $\mathcal{G}^{\mathsf{sat}}_{\mathsf{tb}}(\chi)$ and $n := |\chi|$. There are $1 \le i_1, \dots, i_n \le m$ s.t. for all $j \in \{1, \dots, m\}$ there is a $j' \in \{i_1, \dots, i_n\}$ s.t. γ_j and $\gamma_{j'}$ differ at most in their heads.*

Consequently, the m stacks in a configuration can only have been grown out of n different ones in the previous configuration. This suggests a compact representation of a configuration $\Phi = \gamma_1, \dots, \gamma_m$ with $\gamma_i = \psi_{i,h}; \dots; \psi_{i,0}$ as a single stack over the stack alphabet $\Gamma_\chi = 2^{[n]\times[n]}$. Assume some enumeration $\varphi_0, \dots, \varphi_{n-1}$ of $\mathit{Sub}(\chi)$. Then Φ can be represented as $g_h; \dots; g_1$ where, for each $i = h, \dots, 1$: $(j, k) \in g_i$ if there is j' s.t. $\psi_{j',i} = \varphi_j$ and $\psi_{j',i-1} = \varphi_k$. So in order to reconstruct the stack γ_i, one starts with the index of its head and follows the connections through $g_h; \dots; g_1$ from left to right. This representation technique is also used in the determinisation of VPA [2] and in decision procedures [8].

To overcome the second obstacle of matching $\mathbf{V}$'s winning condition as a stair parity condition, we employ automata-theory. We define a new such alphabet

$$\Sigma^\chi := \{\mathsf{ch}_f \mid f \in \mathit{Sub}^\vee(\chi) \to \{0,1\}\} \cup \{\mathsf{mod}_{\langle a\rangle} \mid \langle a\rangle \in \mathit{Sub}(\chi)\}$$

s.t. $\mathsf{ch}_f \in \Sigma^\chi_{\mathsf{int}}$ for all f, and $\mathsf{mod}_{\langle a\rangle} \in \Sigma^\chi_x$ if $a \in \Sigma_x$. Note that any play of $\mathcal{G}^{\mathsf{sat}}_{\mathsf{tb}}(\chi)$ can easily be represented by a Σ'^ω-word of the form $\mathsf{ch}_{f_0}, \mathsf{mod}_{\langle a_0\rangle}, \mathsf{ch}_{f_1}, \mathsf{mod}_{\langle a_1\rangle}, \dots$. The Σ^χ-symbols uniquely determine the players' alternating choices and, vice-versa, given such a symbolic representation of a play and the initial configuration, all others can be reconstructed uniquely.

▶ **Lemma 21.** *There is a VPA $\mathcal{A}^{\mathsf{thr}}_\chi$ over Σ_χ of size $\mathcal{O}(|\chi|)$ which accepts exactly those (symbolic representations of) plays which contain a μ-thread.*

Proof. $\mathcal{A}^{\mathsf{thr}}_\chi$ stores the current head of a stack in its state, starting with χ. Its stack alphabet is $\mathit{Sub}(\chi) \times \{0,1\}$. Upon reading symbols from Σ^χ, it guesses which thread to follow in the play encoded in the input word and maintains its stack accordingly. The additional bit in its stack symbols is used to guess which positions are stair positions. It also guesses the outermost μ-variable X which will presumably be seen infinitely often in stair positions. Final states are those in which this variable is seen as the head of the stack it follows, when the automaton's internal stack shows that the current position is a stair position. $\mathcal{A}^{\mathsf{thr}}_\chi$ fails immediately, when it should pop a symbol from its internal stack which was marked to be pushed in a stair position, and when a ν-variable Y is seen in a stair position s.t. $X \prec_\chi Y$. ◀

Putting all this together gives an upper bound for vpFLC's satisfiability problem matching the doubly exponential lower bound from Cor. 11.

▶ **Corollary 22.** *The satisfiability problem for* vpFLC *is decidable in* 2EXPTIME*.*

Proof. Thms. 14, 15, Lemma 18 constitute an exponential reduction from general satisfiability games for vpFLC formulas to turn-based ones. Lemmas 19 and 20 show that the arena of these turn-based games is a visibly pushdown frame. Lemma 21 states that the winning condition for player $\mathbf{R}$, i.e. the complement of that of player $\mathbf{V}$ can be recognised by a VPA. The statement then follows from Cor. 6. ◀

5 Conclusion and Further Work

We have presented a decidable fragment of the otherwise undecidable modal fixpoint logic FLC. It makes use of visibly-pushdown principles in order to achieve decidability. However, these principles are built into the logic syntactically and mixed with modal operators unlike similar program logics like PDL[VPL] where the visibly-pushdown principles are separated from the modal part of the logic in the form of so-called (recursive) programs. Consequently, a logic like PDL[VPL] can only make use of such visibly-pushdown principles in expressing properties of some or all paths. Unlike that, in vpFLC these principles can be used to form genuine branching properties. This extends the expressive power notably, making vpFLC a strict superlogic of both PDL[VPL] and the modal μ-calculus and thus pushing the decidability border amongst modal fixpoint logics further up.

The strictness of this extension from PDL[VPL] to vpFLC is a simple consequence of the fact that PDL[VPL] cannot express every $\mathcal{L}_\mu$-definable property. Hence, we have explicit witnesses for the strictness of this inclusion in the form of regular properties like the one mentioned in the introduction about winning two-player games and requiring an unbounded nesting of diamond- and box-formulas in a PDL-like logic. We believe that there are also non-regular properties separating vpFLC from PDL[VPL], for instance the $\langle a\rangle^n[c]\langle b\rangle^n$-property mentioned in the introduction. A detailed study classifying what properties are expressible in vpFLC but not even in PDL[CFL] is left for further research.

There is no formal separation of vpFLC from general FLC in expressive power other than the strong indication given by the decidability border between these two. We strongly believe that vpFLC is in fact less expressive than FLC, and that something like "*there is an n s.t. $\langle a\rangle^n\langle b\rangle\langle a\rangle^n p$ holds*" is not expressible in the former (while it is an easy exercise to formulate it in FLC). Again, in order to prove this formally, techniques need to be developed that better capture the expressive power of such expressive modal fixpoint logics. As a starting point, it seems not to be too difficult to separate vpFLC from FLC over the class of infinite words where the former may only be able to express visibly-pushdown word languages, while the latter can express all properties from the Boolean closure of ω-CFLs [18].

It is possible, albeit technically tedious, to encode multiple stacks in one when they are k-aligned for some $k > 1$. This would allow the syntax of vpFLC to be a bit more relaxed, enabling more formulas to be specified, without breaking the general proof structure. For instance, it may be possible to include τ in the syntax of vpFLC as it can be useful to specify particular properties like "*there is an $n \geq 0$ s.t. $\langle a\rangle^n[b]^n p$ holds*" via $(\mu X.\tau \vee \langle a\rangle; X; [b]); p$. There is no reason why the expressibility of such properties should break decidability, but the free use of τ would require a stronger version of Lemma 19 for the decidability proof to still be valid.

References

1 E. Alsmann, F. Bruse, and M. Lange. Separating the expressive power of propositional dynamic and modal fixpoint logics, 2021. Submitted.

2 R. Alur and P. Madhusudan. Visibly pushdown languages. In *Proc. 36th Annual ACM Symp. on Theory of Computing, STOC'04*, pages 202–211. ACM, 2004. doi:10.1145/1007352.1007390.

3 J. Baran and H. Barringer. A grammatical representation of visibly pushdown languages. In *Proc. 14th Int. Workshop on Logic, Language, Information and Computation, WoLLIC'07*, volume 4576 of *LNCS*, pages 1–11. Springer, 2007. doi:10.1007/978-3-540-73445-1_1.

4 C. S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *Proc. 49th Annual ACM SIGACT Symp. on Theory of Computing, STOC'17*, pages 252–263. ACM, 2017. doi:10.1145/3055399.3055409.

5 E. A. Emerson and C. S. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd Symp. on Foundations of Computer Science*, pages 368–377, San Juan, Puerto Rico, 1991. IEEE. doi:10.1109/sfcs.1991.185392.

6 E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs. *SIAM Journal on Computing*, 29(1):132–158, 2000. doi:10.1137/s0097539793304741.

7 M. J. Fischer and R. E. Ladner. Propositional dynamic logic of regular programs. *Journal of Computer and System Sciences*, 18(2):194–211, 1979. doi:10.1016/0022-0000(79)90046-1.

8 O. Friedmann, F. Klaedtke, and M. Lange. Ramsey-based inclusion checking for visibly pushdown automata. *ACM Trans. Comput. Log.*, 16(4):34, 2015. doi:10.1145/2774221.

9 O. Friedmann and M. Lange. The modal μ-calculus caught off guard. In *20th Int. Conf. on Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX'11*, volume 6793 of *LNCS*, pages 149–163. Springer, 2011. doi:10.1007/978-3-642-22119-4_13.

10 D. Harel, A. Pnueli, and J. Stavi. Propositional dynamic logic of nonregular programs. *Journal of Computer and System Sciences*, 26(2):222–243, 1983. doi:10.1016/0022-0000(83)90014-4.

11 D. Harel and D. Raz. Deciding properties of nonregular programs (preliminary version). In *Proc. 31st Annual Symp. on Foundations of Computer Science, FOCS'90*, volume II, pages 652–661, St. Louis, Missouri, 1990. IEEE. doi:10.1109/fscs.1990.89587.

12 D. Janin and I. Walukiewicz. On the expressive completeness of the propositional μ-calculus with respect to monadic second order logic. In *CONCUR*, pages 263–277, 1996. doi:10.1007/3-540-61604-7_60.

13 M. Jurdziński. Small progress measures for solving parity games. In H. Reichel and S. Tison, editors, *Proc. 17th Ann. Symp. on Theoretical Aspects of Computer Science, STACS'00*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000. doi:10.1007/3-540-46541-3_24.

14 T. Koren and A. Pnueli. There exist decidable context free propositional dynamic logics. In E. Clarke and D. Kozen, editors, *Proc. of the Workshop on Logics of Programs*, volume 164 of *LNCS*, pages 290–312. Springer, 1983. doi:10.1007/3-540-12896-4_369.

15 D. Kozen. Results on the propositional μ-calculus. *TCS*, 27:333–354, 1983. doi:10.1007/BFb0012782.

16 D. Kozen and R. Parikh. A decision procedure for the propositional μ-calculus. In *Proc. Workshop on Logics of Programs*, volume 164 of *LNCS*, pages 313–325. Springer, 1983. doi:10.1007/3-540-12896-4_370.

17 M. Lange. Local model checking games for fixed point logic with chop. In *Proc. 13th Conf. on Concurrency Theory, CONCUR'02*, volume 2421 of *LNCS*, pages 240–254. Springer, 2002. doi:10.1007/3-540-45694-5_17.

18 M. Lange. Temporal logics beyond regularity, 2007. Habilitation thesis, University of Munich, BRICS research report RS-07-13. doi:10.7146/brics.v14i13.22178.

19 M. Lange and R. Somla. Propositional dynamic logic of context-free programs and fixpoint logic with chop. *Information Processing Letters*, 100(2):72–75, 2006. doi:10.1016/j.ipl.2006.04.019.

20 C. Löding, C. Lutz, and O. Serre. Propositional dynamic logic with recursive programs. *J. Log. Algebr. Program*, 73(1-2):51–69, 2007. doi:10.1016/j.jlap.2006.11.003.

21 C. Löding, P. Madhusudan, and O. Serre. Visibly pushdown games. In *Proc. 24th Int. Conf. on Foundations of Software Technology and Theoretical Computer Science, FSTTCS'04*, volume 3328 of *LNCS*, pages 408–420. Springer, 2004. doi:10.1007/978-3-540-30538-5_34.

22 A.W. Mostowski. Games with forbidden positions. Technical Report 78, Uniwersytet Gdański. Instytut Matematyki, 1991.

23 M. Müller-Olm. A modal fixpoint logic with chop. In *Proc. 16th Symp. on Theoretical Aspects of Computer Science, STACS'99*, volume 1563 of *LNCS*, pages 510–520. Springer, 1999. doi:10.1007/3-540-49116-3_48.

24 D. Niwiński and I. Walukiewicz. Games for the μ-calulus. *TCS*, 163:99–116, 1997. doi:10.1016/0304-3975(95)00136-0.

25 R. S. Streett. Propositional dynamic logic of looping and converse. In *Proc. 13th Symp. on Theory of Computation, STOC'81*, pages 375–383. ACM, 1981. doi:10.1145/800076.802492.

26 R. S. Streett and E. A. Emerson. An automata theoretic decision procedure for the propositional μ-calculus. *Information and Computation*, 81(3):249–264, 1989. doi:10.1016/0890-5401(89)90031-x.

27 A. Tarski. A lattice-theoretical fixpoint theorem and its application. *Pacific Journal of Mathematics*, 5:285–309, 1955. doi:10.2140/pjm.1955.5.285.

A PDL over Recursive Programs

We give a brief overview of PDL[VPL]. The underlying action set is a visibly pushdown alphabet $\Sigma = \Sigma_{\mathsf{int}} \uplus \Sigma_{\mathsf{call}} \uplus \Sigma_{\mathsf{ret}}$. Formulas are built as in modal logic, with modalities over VPLs:

$$\varphi \quad ::= \quad q \mid \varphi \wedge \varphi \mid \neg\varphi \mid \langle L\rangle\varphi$$

where $q \in \mathcal{P}$ and L is a (finite representation, for instance in the form of a VPA, of) a VPL. By introducing $\vee$ and $[\cdot]\cdot$ as primitives, formulas can also be given in negation normal form.

Formulas of PDL[VPL] are interpreted over states s of an LTS $\mathcal{T} = (\mathcal{S}, \rightarrow, s_0, \lambda)$ in the following way. First we extend the transition relation $\rightarrow \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ to a path relation $\rightarrow \subseteq \mathcal{S} \times \Sigma^* \times \mathcal{S}$ connecting states by words via

- $s \xrightarrow{\varepsilon} s$ for any $s \in \mathcal{S}$, and
- $s \xrightarrow{aw} t$ if there is $u \in \mathcal{S}$ s.t. $s \xrightarrow{a} u$ and $u \xrightarrow{w} t$, for $s, t \in \mathcal{S}$.

Then the semantics can be given as follows.

$$\begin{array}{rcl}
\mathcal{T}, s \models q & \text{iff} & q \in \lambda(s) \\
\mathcal{T}, s \models \varphi \wedge \psi & \text{iff} & \mathcal{T}, s \models \varphi \text{ and } \mathcal{T}, s \models \psi \\
\mathcal{T}, s \models \neg\varphi & \text{iff} & \mathcal{T}, s \not\models \varphi \\
\mathcal{T}, s \models \langle L\rangle\varphi & \text{iff} & \text{there is } t \in \mathcal{S} \text{ s.t. } s \xrightarrow{w} t \text{ for some } w \in L \text{ and } \mathcal{T}, t \models \varphi
\end{array}$$

PDL[VPL] can then be used to formalise nested visibly-pushdown path properties like $[(L_1)^*]\langle L_1\rangle\mathtt{tt}$ with $L_1 = \{a^n b^n \mid n \geq 1\}$ as mentioned in the introduction. This formula requires every state that is reachable under a (possibly empty) sequence of words from L_1 to be the source of a path with labels from L_1 again. In particular, it implies (but is not equivalent to) the existence of an infinite path with labels of the form $a^{n_1} b^{n_1} a^{n_2} b^{n_2} \ldots$ for some $n_i \geq i$, $i = 1, \ldots$

B Model Checking Games for FLC

We briefly present the essentials of the model checking games for FLC defined in [17] used in the proofs of Thms. 14 and 15.

Given an LTS $\mathcal{T} = (\mathcal{S}, \rightarrow, s_0, \lambda)$ and an FLC-formula χ, the model checking game $\mathcal{G}^{\mathsf{mc}}_{\mathcal{T}}(\chi)$ is played by players $\mathbf{V}$ and $\mathbf{R}$ on configurations of the form $s \vdash \gamma$ where γ is a *stack* of subformulas of χ just like those used in the satisfiability games. The game starts in the initial configuration $s_0 \vdash \chi; \mathtt{tt}$. The rules are presented in Fig. 4.

Player $\mathbf{V}$ wins a finite play $s_0 \vdash \varphi_0; \gamma_0,\ s_1 \vdash \varphi_1; \gamma_1,\ \ldots,\ s_n \vdash \varphi_n; \gamma_n$ if

- $\varphi_n = \mathtt{tt}$,
- $\varphi_n = q$ for some $q \in \mathcal{P}$ and $s_n \in \lambda(q)$,
- $\varphi_n = \overline{q}$ for some $q \in \mathcal{P}$ and $s_n \notin \lambda(q)$,
- $\varphi_n = [a]$ for some $a \in \Sigma$ and there is no t s.t. $s_n \xrightarrow{a} t$.

Player $\mathbf{R}$ wins a finite play $s_0 \vdash \varphi_0; \gamma_0,\ s_1 \vdash \varphi_1; \gamma_1,\ \ldots,\ s_n \vdash \varphi_n; \gamma_n$ if

- $\varphi_n = \mathtt{ff}$,
- $\varphi_n = q$ for some $q \in \mathcal{P}$ and $s_n \notin \lambda(q)$,
- $\varphi_n = \overline{q}$ for some $q \in \mathcal{P}$ and $s_n \in \lambda(q)$,
- $\varphi_n = \langle a\rangle$ for some $a \in \Sigma$ and there is no t s.t. $s_n \xrightarrow{a} t$.

$$\frac{s \vdash (\varphi_0 \vee \varphi_1);\gamma}{s \vdash \varphi_i;\gamma}\ \mathbf{V} : i \in \{0,1\} \qquad \frac{s \vdash (\varphi_0 \wedge \varphi_1);\gamma}{s \vdash \varphi_i;\gamma}\ \mathbf{R} : i \in \{0,1\} \qquad \frac{s \vdash (\varphi_0;\varphi_1);\gamma}{s \vdash \varphi_0;\varphi_1;\gamma}$$

$$\frac{s \vdash \tau;\gamma}{s \vdash \gamma} \qquad \frac{s \vdash \sigma X.\varphi;\gamma}{s \vdash X;\gamma} \qquad \frac{s \vdash X;\gamma}{s \vdash \varphi;\gamma}\ \text{if } \mathit{fp}_\chi(X) = \sigma X.\varphi$$

$$\frac{s \vdash \langle a \rangle;\gamma}{t \vdash \gamma}\ \mathbf{V} : s \xrightarrow{a} t \qquad \frac{s \vdash [a];\gamma}{t \vdash \gamma}\ \mathbf{R} : s \xrightarrow{a} t$$

Figure 4 The rules of the FLC model checking games.

Additionally, the winner of an infinite play $s_0 \vdash \varphi_0;\gamma_0,\ s_1 \vdash \varphi_1;\gamma_1,\ \dots$ is determined by the outermost variable X occurring infinitely often amongst the sequence $(\varphi_{i_j})_{j\geq 0}$ where $i_0, i_1, \dots$ is the sequence of stair positions of the sequence of stacks $\gamma_0, \gamma_1, \dots$[1] If this variable X is of fixpoint type ν, then $\mathbf{V}$ wins this play. Otherwise, if it is type μ, $\mathbf{R}$ wins the play.

▶ **Proposition 23** ([17]). *Let $\mathcal{T}$ be an LTS and χ be a closed* FLC *formula. Player* $\mathbf{V}$ *wins* $\mathcal{G}^{\mathsf{mc}}_{\mathcal{T}}(\chi)$ *iff* $\mathcal{T} \models \chi$.

[1] In [17], this variable is called *stack-increasing*. The terminology of stair positions was coined later in [21].

A Temporal Logic for Strategic Hyperproperties

Raven Beutner
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Bernd Finkbeiner
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Abstract

Hyperproperties are commonly used in computer security to define information-flow policies and other requirements that reason about the relationship between multiple computations. In this paper, we study a novel class of hyperproperties where the individual computation paths are chosen by the strategic choices of a coalition of agents in a multi-agent system. We introduce HyperATL*, an extension of computation tree logic with path variables and strategy quantifiers. HyperATL* can express strategic hyperproperties, such as that the scheduler in a concurrent system has a *strategy* to avoid information leakage. HyperATL* is particularly useful to specify *asynchronous* hyperproperties, i.e., hyperproperties where the speed of the execution on the different computation paths depends on the choices of the scheduler. Unlike other recent logics for the specification of asynchronous hyperproperties, our logic is the first to admit decidable model checking for the full logic. We present a model checking algorithm for HyperATL* based on alternating word automata, and show that our algorithm is asymptotically optimal by providing a matching lower bound. We have implemented a prototype model checker for a fragment of HyperATL*, able to check various security properties on small programs.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Verification by model checking

Keywords and phrases hyperproperties, temporal logic, alternating-time temporal logic, model checking, multi-agent systems, information flow, asynchronous hyperproperties

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.24

Related Version *Full Version*: https://arxiv.org/abs/2107.02509 [6]

Funding This work was partially supported by the German Research Foundation (DFG) as part of the Collaborative Research Center "Foundations of Perspicuous Software Systems" (TRR 248, 389792660), by the European Research Council (ERC) Grant OSARES (No. 683300) and by the Saarbrücken Graduate School of Computer Science.

1 Introduction

Hyperproperties [10] are system properties that specify a relation between the traces of the system. Such properties are of increasing importance as they can, for example, characterize the information-flow in a system [38]. Consequently, several logics for the specification of hyperproperties have been developed, including hyper variants of CTL*(and LTL) [9, 38], PDL-Δ [24] and QPTL [18]. A prominent example is the temporal hyperlogic HyperLTL [9], which extends linear-time temporal logic (LTL) [35] with explicit trace quantification. In HyperLTL we can, for instance, express non-interference (NI), i.e., the requirement that the observable output of a system does not depend on high-security inputs [23]. A prominent formulation of NI for non-deterministic systems is *generalized non-interference* (GNI) [31, 12], which can be expressed as the HyperLTL formula

$$\forall \pi_1 .\ \forall \pi_2 .\ \exists \pi_3 .\ \Box(\bigwedge_{a \in H} a_{\pi_1} \leftrightarrow a_{\pi_3}) \land \Box(\bigwedge_{a \in O} a_{\pi_2} \leftrightarrow a_{\pi_3}),$$

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 24; pp. 24:1–24:19
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

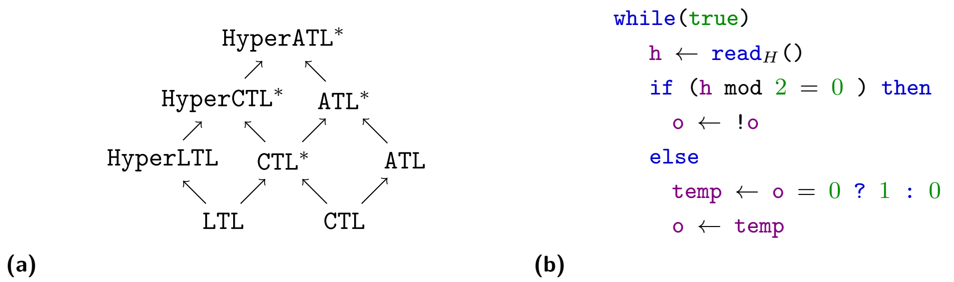
where H and O are two sets of propositions, with H representing the high-security input and O the output. The formula states that for any pair of traces π_1, π_2 there exists a third trace that agrees on the high-security inputs with π_1 and on the outputs with π_2 (for simplicity we assume that no low-security inputs are present). The existence of such a trace guarantees that any observation made on the outputs is compatible with every possible sequence of high-security inputs. The non-determinism is thus the sole explanation for the system output.

In this paper, we introduce a novel class of hyperproperties that reason about *strategic behavior* in a multi-agent system. As a motivation for why strategic hyperproperties are desirable, consider *GNI* from above. As `HyperLTL` only quantifies existentially or universally over the paths in the system, the entire system is treated either as fully controllable or fully adversarial. Moreover, the witness trace π_3 can be constructed with full knowledge of both π_1 and π_2; this means that the entire output and input history can be used to resolve the non-determinism of the system appropriately. Now consider a system where the non-determinism arises from a scheduling decision between two possible subprograms P_1, P_2. Each subprogram reads the next input `h` of the system. Suppose that P_1 assumes that `h` is even and otherwise leaks information, while P_2 assumes that `h` is odd and otherwise leaks information. In the trace-based view of *GNI*, the witness trace π_3 is fixed knowing the entire *future* input sequence, allowing the construction of a leakage-avoiding path π_3; The system satisfies *GNI*. An *actual* scheduler, who chooses which of P_1, P_2 handles the next input, can only avoid a leakage if it knows what the *next* input will be, which is impossible in a real-world system. The `HyperLTL` formulation of *GNI* is, in this case, unable to express the desired property. In our scenario, we need to reason about the *strategic* behaviour of the system, i.e., we want to check if there exist a strategy for the scheduler that avoids leakage.

Strategic Hyperproperties. Reasoning about strategic behavior in multi-agent systems has been studied before. The seminal work on alternating-time temporal logic [1] introduced an extension of `CTL` (and `CTL`*[14]) that is centred around the idea of viewing paths as the outcome of a game, where some agents are controlled via a strategy. The `ATL`* quantifier $\langle\langle A\rangle\rangle\varphi$ requires the agents in A to have a strategy that enforces the path formula φ to become true. This makes `ATL`* an ideal logic for reasoning about *open* systems, where one is less interested in the pure existence of a path, but rather in the actual realizability of an outcome in a multi-agent system. `ATL` has numerous variations and extensions, which, for example, introduce knowledge modalities [42] or imperfect observation [5]. While strategy quantifiers in `ATL`* can be nested (like in `CTL`*), the logic is still unable to express hyperproperties, as the scope of each quantifier ends with the beginning of the next (see [16]).

It is very useful to reason about the strategic behaviour of the agents in a multi-agent system with respect to a hyperproperty. In the example above, one would like to ask if the scheduler has a *strategy* (based on the finite history of inputs only) such that unintended information-flow (which is a hyperproperty) is prevented (in the above example such an answer should be negative). There exist multiple angles to approach this: One could, for instance, interpret strategic hyperproperties such that a coalition of agents tries to achieve a set of outcomes satisfying some hyperproperty (expressed, for example, in `HyperLTL`). Model checking the resulting logic would then subsume realizability of `HyperLTL`, which is *undecidable* even for simple alternation-free formulas [19].

In this paper, we introduce a new temporal logic, called `HyperATL`*, that combines the strategic behaviour in multi-agent systems with the ability to express hyperproperties. Crucially, we focus on the strategic behaviour of a coalition of agents along a *single path*, i.e., we view path quantification as the outcome of a game. Syntactically, we follow a similar



```
while(true)
  h ← read_H()
  if (h mod 2 = 0 ) then
    o ← !o
  else
    temp ← o = 0 ? 1 : 0
    o ← temp
```

Figure 1 (**a**): Expressiveness of temporal logics. An arrow $A \to B$ indicates that A is a syntactic fragment of B. (**b**): Example program that violates (synchronous) observational-determinism.

approach as alternating-time temporal logic [1]. We use the strategy quantifier $\langle\langle A \rangle\rangle \pi.\varphi$ to specify that the agents in A have a strategy such that each possible outcome, when bound to the path variable π, satisfies φ. A formula of the form $\langle\langle A_1 \rangle\rangle \pi_1.\langle\langle A_2 \rangle\rangle \pi_2.\varphi$ now requires the existence of strategy for the agents in A_1 such that for all possible outcomes of the game π_1, the agents in A_2 have a strategy such that for all possible outcomes π_2, the combination of π_1, π_2 satisfies φ (which is a formula that can refer to propositions on paths π_1, π_2). The strategic behaviour chosen by each quantifier is thus limited to the current path and can be based on the already *fixed* outcomes of outer quantifiers (i.e., the entire strategy for the agents in A_2 can depend on the full outcome of π_1). Sometimes, however, it is useful not to reason incrementally about the strategy for a single path at a time, but rather to reason about a *joint* strategy for multiple paths. To express this, we endow our logic with an explicit construct to resolve the games in parallel (syntactically, we surround quantifiers by $[\cdot]$ brackets). The formula $[\langle\langle A_1 \rangle\rangle \pi_1.\langle\langle A_2 \rangle\rangle \pi_2.]\ \varphi$ requires winning strategies for the agents in A_1 (for the first copy) and for A_2 (for the second copy) where the strategies can observe the current state of *both* copies. This enables collaboration between the agents in A_1 and A_2.

Similar to ATL*, the empty (resp. full) agent set corresponds to universal (resp. existential) quantification. HyperATL* therefore subsumes HyperCTL* (and thus HyperLTL) as well as ATL*. The logic is thus a natural extension of both the temporal logics for hyperproperties and the alternating-time logics from the non-hyper realm (see Fig. 1a).

Strategic Non-Interference. Consider again the example of *GNI* expressed in HyperLTL. In HyperATL*, we can express a more refined, strategic notion of non-interference, that postulates the existence of a *strategy* for the non-determinism. As a first step, we consider a program no longer as a Kripke structure (a standard model for temporal hyperlogics), but as a game structure played between two players. Player ξ_N is responsible for resolving the non-determinism of the system, and player ξ_H is responsible for choosing the high-security inputs to the system. We can now express that ξ_N has a strategy to produce matching outputs (without knowing the future inputs by ξ_H). Consider the following formula *stratNI*:

$$\forall \pi_1.\ \langle\langle \{\xi_N\} \rangle\rangle \pi_2.\ \Box\Big(\bigwedge_{a \in O} a_{\pi_1} \leftrightarrow a_{\pi_2}\Big)$$

This formula requires that for every possible reference path π_1, the non-determinism always has a *strategy* to produce identical outputs. One can show that *stratNI* implies *GNI*: The existence of a leakage "disproving" strategy implies the existence of a leakage "disproving" trace. A particular strength of this formulation is that we can encode additional requirements on the strategy. For example: if the internal non-determinism arises from the scheduling decisions between multiple components, we can require fairness of the scheduling strategy.

Asynchronous Hyperproperties. Strategic hyperproperties are also very natural to express *asynchronous hyperproperties.* While existing hyperlogics traverse the traces of a system synchronously, one often requires an asynchronous traversal to account, for example, for the unknown speed of execution of software that runs on some unknown platform. In a multi-agent system, the scheduling decision (i.e., whether a system progresses or remains in its current state) can then be seen as the decision made by scheduling agent (called *sched* in the following). If not already present, we can artificially add such a scheduling agent via a system transformation. By either including or excluding this agent in a strategy quantifier, we can then naturally reason about asynchronous executions of programs. Instead of reasoning about the asynchronous scheduling of a system directly, we thus reason about the existence of a strategy for the scheduling agent.

As an example consider the program in Fig. 1b, which continuously reads an input and flips the output `o` either directly, or via a temporary variable. Based on the input, the exact time point of the change in `o` differs. A synchronous formulation of *observational-determinism* (OD) [26], which requires the output to be identical on all traces, does not hold. In `HyperATL`*, we can naturally express a variant of OD where we search for a strategy for the scheduling agent *sched*, who aligns the outputs on both traces by stuttering them appropriately:

$$[\langle\!\langle \{sched\} \rangle\!\rangle \pi_1.\ \langle\!\langle \{sched\} \rangle\!\rangle \pi_2.]\ \square(\bigwedge_{a \in O} a_{\pi_1} \leftrightarrow a_{\pi_2})$$

The program in Fig. 1b (with an added asynchronous scheduler) satisfies this variant, because *sched* can stutter the change in `o` in order to align with the second trace.

To demonstrate the expressiveness of this strategic view on asynchronous hyperproperties, we compare our approach to `AHLTL`, a recent temporal logic for asynchronous hyperproperties [4]. While `AHLTL` model checking is undecidable in general, recent work [4] has identified a large fragment for which model checking is possible. We show that this fragment can be encoded within `HyperATL`*. Every property in this (largest known) decidable fragment can thus be expressed in `HyperATL`*, for which model checking is decidable for the *full* logic.

Model Checking. We show that model checking of `HyperATL`* on concurrent game structures is decidable and present an automata-theoretic algorithm. Our algorithm incrementally reduces model checking to the emptiness of an automaton. We show that alternating automata are well suited to keep track of all possible path assignments satisfying a formula by encoding the game structure in the transition function of the automaton. We characterize the model checking complexity in terms of the number of complex quantifiers (where the agent team is non-trivial) and simple quantifiers (i.e., $\exists$ or $\forall$). We provide a lower bound, based on a novel construction that encodes a doubly exponential counter within a single strategy quantifier, that (in almost all cases) matches the upper bound from our algorithm.

Prototype Model Checker. On the practical side, we present a prototype model checker for an efficient fragment of `HyperATL`* by reducing the model checking to solving of a parity game. The fragment supported by our tool does, in particular, include all alternation free `HyperLTL` formulas [20], the $\forall^*\exists^*$-model checking approach from [12] as well as all formulas in the decidable fragment of `AHLTL` [4].

Contributions. In summary, our contributions include the following:

- We introduce a novel logic to express strategic hyperproperties and demonstrate that it is well suited to express, e.g., information-flow control and, in particular, asynchronous hyperproperties.

- We give an automata-based model checking algorithm for our logic and provide a lower bound on the model checking problem.
- We show that our logic can express all formulas in the largest known decidable fragment of the existing hyperlogic `AHLTL` [4].
- We provide a prototype-model checker for an efficiently checkable fragment of `HyperATL`* and use it to verify information-flow polices and asynchronous hyperproperties.

2 Preliminaries

In this section, we introduce some basic preliminaries needed in the following.

Concurrent Game Structure. As our model of multi-agent systems, we consider concurrent game structures (CGS) [1]. The transition relation in a CGS is based on the decision by individual agents (or players). Formally, a CGS is a tuple $\mathcal{G} = (S, s_0, \Xi, \mathcal{M}, \delta, \mathbf{AP}, L)$ where S is a finite set of states, $s_0 \in S$ the initial state, Ξ a finite set of agents and $\mathcal{M}$ a finite set of moves. We call a function $\sigma : \Xi \to \mathcal{M}$ a global move vector and for a set of agent $A \subseteq \Xi$ a function $\sigma : A \to \mathcal{M}$ a partial move vector. $\delta : S \times (\Xi \to \mathcal{M}) \to S$ is a transition function that maps states and move vectors to successor states. Finally, $\mathbf{AP}$ is a finite set of propositions and $L : S \to 2^{\mathbf{AP}}$ a labelling function. Note that every Kripke structure (a standard model for temporal logics [3]) can be seen as a 1-player CGS. For disjoint sets of agents A_1, A_2 and partial move vectors $\sigma_i : A_i \to \mathcal{M}$ for $i \in \{1, 2\}$ we define $\sigma_1 + \sigma_2$ as the move vector obtained as the combination of the individual choices. For $\sigma : A \to \mathcal{M}$ and $A' \subseteq A$, we define $\sigma_{|A'}$ as the move vector obtained by restring the domain of σ to A'.

In a concurrent game structure (as the name suggests) all agents choose their next move concurrently, i.e., without knowing what moves the other player have chosen. We introduce the concept of multi-stage CGS (MSCGS), in which the move selection proceeds in stages and agents can base their decision on the already selected moves of (some of the) other agents. This is particularly useful when we, e.g., want to base a scheduling decision on the moves selected by the other agents. Formally, a MSCGS is a CGS equipped with a function $d : \Xi \to \mathbb{N}$, that orders the agents according to informedness. Whenever $d(\xi_1) < d(\xi_2)$, ξ_2 can base its next move on the move selected by ξ_1. A CGS thus naturally corresponds to a MSCGS with $d = \mathbf{0}$, where $\mathbf{0}$ is the constant 0 function.

Alternating Automata. An alternating parity (word) automaton (APA) is a tuple $\mathcal{A} = (Q, q_0, \Sigma, \rho, c)$ where Q is a finite set of states, $q_0 \in Q$ an initial state, Σ a finite alphabet, $\rho : Q \times \Sigma \to \mathbb{B}^+(Q)$ a transition function ($\mathbb{B}^+(Q)$ is the set of positive boolean combinations of states) and $c : Q \to \mathbb{N}$ a colouring of nodes with natural numbers. For $\Psi \in \mathbb{B}^+(Q)$, $B \subseteq Q$ we write $B \models \Psi$ if the assignment obtained from B satisfies Ψ. A tree is a set $T \subseteq \mathbb{N}^*$ that is prefixed closed, i.e., $\tau \cdot n \in T$ implies $\tau \in T$. We refer to elements in $\tau \in T$ as nodes and denote with $|\tau|$ the length of τ (or equivalently the depth of the node). For a node $\tau \in T$ we denote with $\mathit{children}(\tau)$ the immediate children of τ, i.e., $\mathit{children}(\tau) = \{\tau \cdot n \in T \mid n \in \mathbb{N}\}$. A X-labelled tree is a pair (T, r) where T is a tree and $r : T \to X$ a labelling with X. A run of an APA $\mathcal{A} = (Q, q_0, \Sigma, \rho, c)$ on a word $u \in \Sigma^\omega$ is a Q-labelled tree (T, r) that satisfies the following: **(1)** $r(\epsilon) = q_0$, **(2)** For all $\tau \in T$, $\{r(\tau') \mid \tau' \in \mathit{children}(\tau)\} \models \rho(r(\tau), u(|\tau|))$. A run (T, r) is accepting if for every infinite path π in T the minimal colour (given by c) that occurs infinitely many times is even. We denote with $\mathcal{L}(\mathcal{A})$ the set of words for which $\mathcal{A}$ has an accepting run. We call an alternating automaton $\mathcal{A}$ non-deterministic (resp. universal) if the transition function δ is a disjunction (resp. conjunction) of states. If δ is just a single state, we call $\mathcal{A}$ deterministic. Crucially alternating, non-deterministic, universal and deterministic parity automaton are all equivalent in the sense that they accept the same class of languages (namely ω-regular ones) although they can be (double) exponentially more succinct:

▶ **Theorem 1** ([33, 13]). *For every alternating parity automaton $\mathcal{A}$ with n states, there exists a non-deterministic parity automaton $\mathcal{A}'$ with $2^{\mathcal{O}(n)}$-states that accepts the same language. For every non-deterministic or universal parity automaton $\mathcal{A}$ with n states, there exists a deterministic parity automaton $\mathcal{A}'$ with $2^{\mathcal{O}(n)}$-states that accepts the same language.*

▶ **Theorem 2** ([29]). *For every alternating parity automaton $\mathcal{A}$ with n states, there exists an alternating parity automaton $\overline{\mathcal{A}}$ with $\mathcal{O}(n^2)$-states that accepts the complemented language.*

3 HyperATL*

In this section, we introduce HyperATL*. Our logic extends the standard temporal logic CTL* [14] by introducing path variables and strategic quantification [1]. Assume a countably infinite set of path variables *Var*, a set of agents Ξ and a set of atomic propositions $\mathbf{AP}$. HyperATL* formulas are generated by the following grammar

$$\varphi := \langle\!\langle A \rangle\!\rangle \pi.\varphi \mid a_\pi \mid \varphi \land \varphi \mid \neg\varphi \mid \varphi \,\mathcal{U}\, \varphi \mid \bigcirc \varphi$$

where $\pi \in \mathit{Var}$ is a path variable, $a \in \mathbf{AP}$ an atomic proposition and $A \subseteq \Xi$ a set of agents. As in HyperCTL*, a_π means that proposition a holds in the current step on path π. Via $\langle\!\langle A \rangle\!\rangle \pi.\varphi$ we can quantify over paths in a system (which we consider as the outcome of a game). $\langle\!\langle A \rangle\!\rangle \pi.\varphi$ requires the agents in A to have a *strategy* (defined below) such that each outcome under that strategy, when bound to trace variable π, satisfies φ. We abbreviate as usual $\varphi_1 \lor \varphi_2 := \neg(\neg\varphi_1 \land \neg\varphi_2)$, and the temporal operators globally ($\square$), eventually ($\diamondsuit$) and release ($\mathcal{R}$). Trivial agent sets, i.e., $A = \emptyset$ or $A = \Xi$ correspond to classical existential or universal quantification. We therefore write $\forall\pi$ instead of $\langle\!\langle \emptyset \rangle\!\rangle \pi$ and $\exists\pi$ instead of $\langle\!\langle \Xi \rangle\!\rangle \pi$. We call a quantifier *simple* if the agent-set is trivial and otherwise *complex.* We call a formula *linear* if it consists of an initial quantifier prefix followed by a quantifier-free (LTL) formula.

Semantics. Let us fix a MSCGS $\mathcal{G} = (S, s_0, \Xi, \mathcal{M}, \delta, d, \mathbf{AP}, L)$ as a model. We first need to formalize the notion of a strategy in the game structure. A strategy for any agent is a function that maps finite histories of plays in the game to a move in $\mathcal{M}$. As the plays in an MSCGS progress in stages, the decision can be based not only on the past sequence of states, but also on the fixed moves of all agents in previous stages. Formally, a strategy for an agent ξ is a function $f_\xi : S^+ \times (\{\xi' \mid d(\xi') < d(\xi)\} \to \mathcal{M}) \to \mathcal{M}$. Given a set of agents A, a set of strategies $F_A = \{f_\xi \mid \xi \in A\}$ and a state $s \in S$, we define $\mathit{out}(\mathcal{G}, s, F_A)$ as the set of all runs $u \in S^\omega$ such that **(1)** $u(0) = s$ and **(2)** for every $i \in \mathbb{N}$ there exists a global move vector σ with $\delta(u(i), \sigma) = u(i+1)$ and for all $\xi \in A$ we have $\sigma(\xi) = f_\xi(u[0,i], \sigma_{|\{\xi' | d(\xi') < d(\xi)\}})$. The agents in A choose their move in each step based on the finite history of the play and the decision of all other agents in an earlier stage. Note that in case where $d = \mathbf{0}$, a strategy is just a function $S^+ \to \mathcal{M}$, ignoring the moves selected by other agents.

The semantics of a formula is now defined in terms of a path assignment $\Pi : \mathit{Var} \to S^\omega$, mapping path variables to infinite sequences of states in $\mathcal{G}$. For a path $t \in S^\omega$ we write $t[i, \infty]$ to refer to the infinite suffix of t starting at position i. We write $\Pi[i, \infty]$ to denote the path assignment defined by $\Pi[i, \infty](\pi) = \Pi(\pi)[i, \infty]$. We can then inductively define the satisfaction relation for HyperATL*:

$$\begin{array}{ll}
\Pi \models_{\mathcal{G}} a_\pi & \text{iff } a \in L(\Pi(\pi)(0)) \\
\Pi \models_{\mathcal{G}} \neg\varphi & \text{iff } \Pi \not\models_{\mathcal{G}} \varphi \\
\Pi \models_{\mathcal{G}} \varphi_1 \land \varphi_2 & \text{iff } \Pi \models_{S} \varphi_1 \text{ and } \Pi \models_{\mathcal{G}} \varphi_2 \\
\Pi \models_{\mathcal{G}} \varphi_1 \,\mathcal{U}\, \varphi_2 & \text{iff } \exists i \geq 0. \Pi[i,\infty] \models_{\mathcal{G}} \varphi_2 \text{ and } \forall 0 \leq j < i. \Pi[j,\infty] \models_{\mathcal{G}} \varphi_1 \\
\Pi \models_{\mathcal{G}} \bigcirc\varphi & \text{iff } \Pi[1,\infty] \models_{\mathcal{G}} \varphi \\
\Pi \models_{\mathcal{G}} \langle\!\langle A \rangle\!\rangle \pi.\ \varphi & \text{iff } \exists F_A : \forall t \in \mathit{out}(\mathcal{G}, \Pi(\epsilon)(0), F_A) : \Pi[\pi \mapsto t] \models_{\mathcal{G}} \varphi
\end{array}$$

Here $\Pi(\epsilon)$ refers to the path that was last added to the assignment (similar to the HyperLTL-semantics [9]). If Π is the empty assignment, we define $\Pi(\epsilon)(0)$ as the initial state s_0 of $\mathcal{G}$. Note that the games are local to each path but based on all outer paths.: In a formula of the form $\forall\pi_1.\langle\!\langle A \rangle\!\rangle\pi_2.\varphi$ the agents in A know the already fixed, *full* trace π_1 but behave as a strategy w.r.t. π_2. We write $\mathcal{G} \models \varphi$ whenever $\emptyset \models_{\mathcal{G}} \varphi$ where $\emptyset$ is the empty path assignment.

▶ **Proposition 3.** *HyperATL* subsumes HyperCTL* (and thus HyperLTL) and ATL* (see Fig. 1a).*

We sometimes consider HyperATL* formulas with extend path quantification: We write $\langle\!\langle A \rangle\!\rangle_{\mathcal{G}}\, \pi.\varphi$ to indicate that the path π is the result of the game played in $\mathcal{G}$. We can thus refer to different structures in the same formula. For example, $\forall_{\mathcal{G}_1}\, \pi_1.\ \langle\!\langle A \rangle\!\rangle_{\mathcal{G}_2}\, \pi_2.\ \square(o_{\pi_1} \leftrightarrow o_{\pi_2})$ states that for each path π_1 in $\mathcal{G}_1$ the agents in A have a strategy in $\mathcal{G}_2$ that produces the same outputs as on π_1. As for HyperLTL, extended quantification reduces to the standard semantics [38, §5.4].

Parallel-Composition. We extend HyperATL* with a *syntactic* construct that allows multiple traces to be resolved in a single bigger game, where individual copies of the system progress in parallel. Consider the following modification to the HyperATL* syntax, where $k \geq 1$:

$$\varphi := [\langle\!\langle A_1 \rangle\!\rangle \pi_1.\ \cdots\ \langle\!\langle A_k \rangle\!\rangle \pi_k.]\ \varphi \mid a_\pi \mid \neg\varphi \mid \varphi \land \varphi \mid \varphi \,\mathcal{U}\, \varphi \mid \bigcirc\varphi$$

When surrounding strategy quantifiers by $[\cdot]$ the resulting traces are the outcome of a game played on a bigger, parallel game of the structure. This way, the agents in each copy can base their decisions not only on the current state of their copy but on the combined state of all k copies (which allows for a coordinated behaviour among the copies). For a player ξ, and a CGS $\mathcal{G} = (S, s_0, \Xi, \mathcal{M}, \delta, \mathbf{AP}, L)$, a k-strategy for ξ is a function $f_\xi : (S^k)^+ \to \mathcal{M}$. The strategy can thus base its decision on a finite history for each of the k copies. For a system $\mathcal{G}$, sets of k-strategies strategies $F_{A_1}, \cdots, F_{A_k}$ and states $s_1, \cdots, s_k$, we define $\mathit{out}(\mathcal{G}, (s_1, \cdots, s_k), F_{A_1}, \cdots, F_{A_k})$ as all plays $u \in (S^k)^\omega$ such that **(1)** $u(0) = (s_1, \cdots, s_k)$ and **(2)** for every $i \in \mathbb{N}$ there exist move vectors $\sigma_1, \cdots, \sigma_k$ such that $u(i+1) = (\delta(t_1, \sigma_1), \cdots, \delta(t_k, \sigma_k))$ where $u(i) = (t_1, \cdots, t_k)$ and for every $j \in \{1, \cdots, k\}$, agent $\xi \in A_j$ and strategy $f_\xi \in F_{A_j}$, $\sigma_j(\xi) = f_\xi(u[0,i])$. Agents can thus control the individual progress of their system and base their decision on the history of the other quantifiers. Note that in the case where $k = 1$ this is identical to the construction seen above. For simplicity, we gave the semantics for a CGS (i.e., a MSCGS without stages), it can be generalized easily. We can now extend our semantics by

$$\begin{array}{l}
\Pi \models_{\mathcal{G}} [\langle\!\langle A_1 \rangle\!\rangle \pi_1.\ \cdots\ \langle\!\langle A_k \rangle\!\rangle \pi_k.]\ \varphi \text{ iff} \\
\quad \exists F_{A_1}, \cdots, F_{A_k} : \forall (t_1, \cdots, t_k) \in \mathit{out}(\mathcal{G}, (\Pi(\epsilon)(0), \cdots, \Pi(\epsilon)(0)), F_{A_1}, \cdots, F_{A_k}) : \Pi[\pi_i \mapsto t_i]_{i=1}^{k} \models_{\mathcal{G}} \varphi
\end{array}$$

Note that $[\langle\!\langle A \rangle\!\rangle \pi.]\ \varphi$ is equivalent to $\langle\!\langle A \rangle\!\rangle \pi.\varphi$. This does, of course, not hold once we consider multiple strategy quantifiers grouped together by $[\cdot]$.

Comparison with ∀∃-HyperLTL model checking [12]. To give some more intuition for the parallel composition, we can compare our $[\cdot]$-construct with the model checking algorithm introduced in [12]. The idea of the method from [12] is to check a ∀∃-formula by viewing the existential quantifier as a player who has to decide on a next state (in her copy) by reacting to the moves of the universal quantifier. If such a winning strategy exists, the ∀∃-formula also holds, whereas the absence of a winning strategy does, in general, *not* imply that the formula is invalid (as the strategy bases its decision on finite plays whereas the existential path is chosen with the universally quantified path already fixed). This game based view of the existential player can be natively expressed in HyperATL*: While the HyperATL*-formula $\forall \pi_1.\exists \pi_2.\varphi$ is equivalent to the same HyperLTL-formula (i.e., the existential trace π_2 is chosen knowing the entire trace π_1), model checking of the formula $[\forall \pi_1.\exists \pi_2].\varphi$ corresponds to the strategy search for the existential player that is only based on finite prefixes of π_1 (which directly corresponds to [12]). We can actually show that if any MSCGS $\mathcal{G}$ satisfies $[\forall \pi_1.\langle\!\langle A \rangle\!\rangle \pi_2.]\varphi$ then it also satisfies $\forall \pi_1.\ \langle\!\langle A \rangle\!\rangle \pi_2.\varphi$ (see [6]). This gives a more general proof of the soundness of [12]. As our prototype implementation supports $[\langle\!\langle A_1 \rangle\!\rangle \pi_1.\langle\!\langle A_2 \rangle\!\rangle \pi_2.]$-formulas, our tool subsumes the algorithm in [12] (see Sec. 8).

4 Examples of Strategic Hyperproperties

After having introduced the formal semantics of HyperATL*, we now demonstrate how the strategic quantification can be useful for expressing hyperproperties. We organize our example in two categories. We begin with examples from information-flow control and highlight the correspondence with existing properties and security paradigms. Afterwards, we show how the strategic hyperproperties are naturally well suited to express asynchronous hyperproperties.

4.1 Strategic Information-Flow Control

We focus our examples on game structures that result from a reactive system. Let H (resp. L) be the set of atomic propositions forming the high-security (resp. low-security) inputs of a system (we assume $H \cap L = \emptyset$). The game structure then comprises 3-players ξ_N, ξ_H, ξ_L, responsible for resolving non-determinism and selecting high-security and low-security inputs. In particular, the move from ξ_H (resp. ξ_L) determines the values of the propositions in H (resp. L) in the next step. We call a system *input-total*, if in each state, ξ_H and ξ_L can choose all possible valuations for the input propositions.

Strategic Non-Interference. In the introduction, we already saw that *GNI* [31] is (in some cases) a too relaxed notion of security, as it can base the existence of a witness trace on knowledge of the entire input-sequence. Note that *GNI* can be extended to also allow for input from a low-security source that may affect the output. The HyperATL* formula *stratNI* (given in the introduction) instead postulates a *strategy* for ξ_N that incrementally constructs a path that “disproves“ information leakage. We can show that *stratNI* implies *GNI*. Loosely speaking, whenever there is a strategy for the non-determinism based on the finite history of inputs, there also exists a path when given the full history of inputs (as in *GNI*).

▶ **Lemma 4.** *For any system $\mathcal{G}$ that is input-total, we have that if $\mathcal{G} \models$ stratNI then $\mathcal{G} \models$ GNI.*

Simulation-based Non-Interference. Other attempts to non-interference are based on the existence of a bisimulation (or simulation) [40, 39, 30]. While trace-based notions of non-interference (such as *GNI*) only require the existence of a path that witnesses the absence of a leak, simulation based properties require a lock-step relation in which this holds. Given a system $\mathcal{G}$ with initial states s_0. For states s, s' and evaluations $i_L \in 2^L$ and $i_H \in 2^H$, we write $s \Rightarrow_{i_H}^{i_L} s'$ if $L(s') \cap L = i_L$ and $L(s') \cap H = i_H$ and s' is a possible successor of s. A *security simulation* is a relation R on the states of S such that whenever sRt, we have **(1)** s and t agree on the output propositions, and **(2)** for any $i_L \in 2^L$ and $i_H, i'_H \in 2^H$ if $s \Rightarrow_{i_H}^{i_L} s'$ then there exists a t' with $t \Rightarrow_{i'_H}^{i_L} t'$ and $s'Rt'$. Note that this is not equivalent to the fact that $\Rightarrow$ is a simulation in the standard sense [32]. While a standard simulation relation is always reflexive, reflexivity of security simulations guarantees the security of the system [39, 40]. A system is thus called *simulation secure* if there exists a security simulation R with s_0Rs_0. It is easy to see that every input-total system that is *simulation secure* already satisfies *GNI*. The converse does, in general, not hold. We can show that `HyperATL`* can express *simulation security* by using the parallel-composition of quantifiers.

$$[\forall_{\mathcal{G}}\, \pi_1.\ \langle\!\langle \{\xi_N\} \rangle\!\rangle_{\mathcal{G}_{shift}}\, \pi_2.]\ \Box\Big(\bigwedge_{a \in L} a_{\pi_1} \leftrightarrow \bigcirc a_{\pi_2}\Big) \rightarrow \Box\Big(\bigwedge_{a \in O} a_{\pi_1} \leftrightarrow \bigcirc a_{\pi_2}\Big)$$

Here we consider `HyperATL`* with extended quantifier, where we annotate each quantifier with the game structure it is resolved on. $\mathcal{G}_{shift}$ is the structure $\mathcal{G}$ where we added a dummy initial state, that shifts the behaviour of the system by one position, which is again corrected via the next operator in the `LTL` formula. This allows the strategy for ξ_N in the second copy to base its decision on an already fixed step in the first copy, i.e., it corresponds to a strategy with a fixed lookahead of 1 step. We can show:

▶ **Lemma 5.** *An input-total system $\mathcal{G}$ is* simulation secure *if and only if it satisfies simNI.*

Non-Deducibility of Strategies. Lastly, we consider the notion of *non-deducibility of strategies* (*NDS*) [45]. *NDS* requires not only that each output is compatible with each sequence of inputs, but also with each input-*strategy*. This becomes important as a high-security input player who can observe the internal state of a system might be able to leak information deliberately. As a motivating example, consider the following (first introduced in [45]): Suppose we have a system that reads a binary input `h` from a high-security source and outputs `o`. The system maintains a bit b of information in its state, initially chosen non-deterministically. In each step, the system reads the input `h`, outputs `h` $\oplus\, b$ (where $\oplus$ is the xor-operation), non-deterministically chooses a new value for b and then repeats. As $\oplus$ essentially encodes a one-time pad it is not hard to see, that this system is secure from a purely trace-based point of view (as expressed in e.g. *GNI*): Any possible combination of input and output can be achieved when resolving the non-deterministic choice of b appropriately. If the high-input player is, however, able to observe the system (in the context of [45] the system shares the internal bit on a private channel), she can communicate arbitrary sequence of bits to the low-security environment. Whenever she wants to send bit c, she inputs $c \oplus b$ where b is the internal bit she has access to (note that $(c \oplus b) \oplus b = c$). For such system system we therefore do no want to specify that every possible output sequence is compatible with all possible inputs, but instead compatible with all possible input-*strategies* (based on the state of the system). Phrased differently, there should *not* be a output sequence such that a strategy can reliably *avoid* this output. We can express *NDS* in `HyperATL`* as follows:

$$\neg\left(\exists \pi_1.\ \langle\!\langle \xi_H \rangle\!\rangle \pi_2.\ \Box\Big(\bigwedge_{a \in L} a_{\pi_1} \leftrightarrow a_{\pi_2}\Big) \rightarrow \Diamond\Big(\bigvee_{a \in O} a_{\pi_1} \not\leftrightarrow a_{\pi_2}\Big)\right)$$

This formula states that there does not exist a trace π_1 such that ξ_H has a strategy to avoid the output of π_1 (provided with the same low-security inputs). *NDS* is a stronger requirement than *GNI*, as shown by the following Lemma:

▶ **Lemma 6.** *For any system $\mathcal{G}$, if $\mathcal{G} \models NDS$ then $\mathcal{G} \models GNI$.*

4.2 Asynchronous Hyperproperties

Reasoning about the strategic behaviour of agents is particularly useful when reasoning about asynchronous hyperproperties, as each asynchronous execution can be considered the result of the decision of an asynchronous scheduler. We call a player an asynchronous scheduler if it can decide whether the system progresses (as decided by the other agents) or stutters. Note that this differs from the asynchronous turn-based games defined in [1]. In our setting, the scheduler does not control which of the player controls the next move, but rather decides if the system as a whole progresses or stutters. In cases where the system does not already include such an asynchronous scheduler (if we e.g. use a Kripke structure interpreted as a 1-player CGS), we can include a scheduler via a simple system transformation:

▶ **Definition 7.** *Given a MSCGS $\mathcal{G} = (Q, q_0, \Xi, \mathcal{M}, \delta, d, \mathbf{AP}, L)$ and a fresh agent* sched *not already included in the set of agents of Ξ. We define the stutter version of $\mathcal{G}$, denoted $\mathcal{G}_{stut}$, by $\mathcal{G}_{stut} := (Q \times \{0,1\}, (q_0, 0), \Xi \uplus \{sched\}, \mathcal{M} \times \{0,1\}, \delta', d', \mathbf{AP} \uplus \{stut\}, L')$ where*

$$\delta'((s,b),\sigma) = \begin{cases} (\delta(s, proj_1 \circ \sigma_{|\Xi}), 0) & \text{if } (proj_2 \circ \sigma)(sched) = 0 \\ (s,1) & \text{if } (proj_2 \circ \sigma)(sched) = 1 \end{cases}$$

$L'((s,0)) = L(s)$ and $L'(s,1) = L(s) \cup \{stut\}$. Finally $d'(\xi) = d(\xi)$ for $\xi \in \Xi$ and $d'(sched) = m+1$ where m is the maximal element in the codomain of d.

Here $proj_i$ is the projection of the ith element in a tuple and $\circ$ denotes function composition, i.e., $(f \circ g)(x) := f(g(x))$. $\mathcal{G}_{stut}$ thus progresses as $\mathcal{G}$ with the exception of the additional scheduling player. In each step, the $\{0,1\}$-decision of *sched*, which can be based on the decision by the other agents (as *sched* is in the last stage of the game), decides if the system progresses or remains in its current state. The extended state-space $Q \times \{0,1\}$ is used to keep track of the stuttering which becomes visible via the new atomic proposition *stut*. Our construction will be particularly useful when comparing our logic to AHLTL [4].

Observational Determinism. As an example we consider observational-determinism which states that the output along all traces is identical (in HyperLTL $OD := \forall \pi_1 . \forall \pi_2 .\ \square(\bigwedge_{a \in O} a_{\pi_1} \leftrightarrow a_{\pi_2})$). The example in Fig. 1b does not satisfy this property, as the output is changed at different timepoints. If we consider any system as a multi-agent system including the scheduler *sched*, we can use HyperATL* to naturally express an asynchronous version of OD via:

$$OD_{asynch} := [\langle\!\langle \{sched\} \rangle\!\rangle \pi_1 . \langle\!\langle \{sched\} \rangle\!\rangle \pi_2 .]\ fair_{\pi_1} \wedge fair_{\pi_2} \wedge \square(\bigwedge_{a \in O} a_{\pi_1} \leftrightarrow a_{\pi_2})$$

where $fair_{\pi_i} := \square \lozenge \neg stut_{\pi_i}$, asserts that the system may not be stuttered forever. Note that we encapsulated the quantifiers by $[\cdot]$ thus resolving the games in parallel. The schedulers for both copies of the system can thus observe the current state of the other copy. The example from Fig. 1b, after a transformation via Definition 7, satisfies this formula, as the output can be aligned by the scheduling player.

One-Sided Stuttering. By resolving the stuttered traces incrementally (i.e., omitting the $[\cdot]$-brackets) we can also express *one-sided stuttering*, i.e., allow only the second copy to be stuttered. As an example assume P^o is a program written in the high-level programming language and P^a the complied program into binary code. Let S^o and S^a be the state systems of both programs. Using HyperATL* we can now verify that the compiler did not

a_{π_i} $\neg a_{\pi_i}$	$\mathcal{A}_\varphi = (\{q_{init}\}, q_{init}, \Sigma_\varphi, \rho, \mathbf{0})$ $\rho(q_{init}, [s_1, \cdots, s_n]) = \begin{cases} \overset{\top}{\underset{\bot}{}} & \text{if } a \in L(s_i) \\ \overset{\bot}{\underset{\top}{}} & \text{if } a \notin L(s_i) \end{cases}$
$\varphi_1 \overset{\vee}{\underset{\wedge}{}} \varphi_2$	$\mathcal{A}_\varphi = (Q_1 \cup Q_2 \cup \{q_{init}\}, q_{init}, \Sigma_\varphi, \rho, c_1 \uplus c_2 \uplus [q_{init} \mapsto 0])$ $\rho(q, [s_1, \cdots, s_n]) = \begin{cases} \rho_1(q_{0,1}, [s_1, \cdots, s_n]) \overset{\vee}{\underset{\wedge}{}} \rho_2(q_{0,2}, [s_1, \cdots, s_n]) & \text{if } q = q_{init} \\ \rho_i(q, [s_1, \cdots, s_n]) & \text{if } q \in Q_i \end{cases}$
$\bigcirc \varphi_1$	$\mathcal{A}_\varphi = (Q_1 \cup \{q_{init}\}, q_{init}, \Sigma_\varphi, \rho, c_1 \uplus [q_{init} \mapsto 0])$ $\rho(q, [s_1, \cdots, s_n]) = \begin{cases} q_{0,1} & \text{if } q = q_{init} \\ \rho_1(q, [s_1, \cdots, s_n]) & \text{if } q \in Q_1 \end{cases}$
$\varphi_1 \overset{\mathcal{U}}{\underset{\mathcal{R}}{}} \varphi_2$	$\mathcal{A}_\varphi = (Q_1 \cup Q_2 \cup \{q_{init}\}, q_{init}, \Sigma_\varphi, \rho, c_1 \uplus c_2 \uplus [q_{init} \mapsto \overset{1}{\underset{0}{}}])$ $\rho(q, [s_1, \cdots, s_n]) = \begin{cases} \rho_2(q_{0,2}, [s_1, \cdots, s_n]) \overset{\vee}{\underset{\wedge}{}} (\rho_1(q_{0,1}, [s_1, \cdots, s_n]) \overset{\wedge}{\underset{\vee}{}} q_{init}) & \text{if } q = q_{init} \\ \rho_i(q, [s_1, \cdots, s_n]) & \text{if } q \in Q_i \end{cases}$

Figure 2 APA construction for LTL temporal operators. $\mathcal{A}_{\varphi_i} = (Q_i, q_{0,i}, \Sigma_{\varphi_i}, \rho_i, c_i)$ is the inductively constructed automaton for φ_i.

leak information, i.e., the assembly code does provide the same outputs as the original code. As the compiler breaks each program statement into multiple assembly instructions, we can not require the steps to match in a synchronous manner. Instead, the system S^o should be allowed to stutter for the assembly program to catch up. We can express this as follows:

$$\forall_{S^a} \pi_1. \langle\!\langle \{sched\} \rangle\!\rangle_{S^o_{stut}} \pi_2. \; fair_{\pi_2} \land \Box(\bigwedge_{a \in O} a_{\pi_1} \leftrightarrow a_{\pi_2})$$

I.e., for every execution of the assembly code we can stutter the program such that the observations align. Here, S^o_{stut} denotes the modified version obtained by introducing an explicit scheduler (Definition 7). Note that we use extend path quantification by annotating a quantifier with the game structure, thereby effectively comparing both systems with respect to a hyperproperty.

5 Automata-Theoretic Model Checking

In this section we present an automata-based algorithm for HyperATL* model checking. The crucial insight in our algorithm is how to deal with the strategic quantification. Let us briefly recall ATL* model checking [1]: In ATL*, checking if $\langle\!\langle A \rangle\!\rangle \varphi$ holds in some state s, can be reduced to the non-emptiness check of the intersection of two tree automata. One accepting all possible trees that can be achieved via a strategy for players in A, and one accepting all trees whose paths satisfy the path formula φ [1]. In our hyperlogic this is not possible. When checking $\langle\!\langle A \rangle\!\rangle \pi.\varphi$ we can not construct an automaton accepting all trees that satisfy φ, as the satisfaction of φ depends on the paths assigned to the outer path-quantifiers (that are not yet fixed). Instead, we construct an automaton that accepts all *path assignments* for the outer quantifiers such that there exists a winning strategy for the agents in A. We show that alternating automata are well suited to keep track of all path assignments for which a strategy exists, as they allow us to encode the strategic behaviour of $\mathcal{G}$ within the transition function of the automaton.

We assume that the formula φ to be checked is given in *negation normal form*, i.e., negations only occur directly in front of atomic propositions or in front of a path quantifier. By including conjunction ($\land$) and release ($\mathcal{R}$) every formula can be translated into a negation normal form of linear size. We, furthermore, assume that if $\langle\!\langle A \rangle\!\rangle \pi.\varphi$ occurs in the formula, we have $A \neq \emptyset$. Note that in this case where $A = \emptyset$ we can use that $\forall \pi.\varphi \equiv \neg \exists \pi. \neg \varphi$. For infinite words $t_1, \cdots, t_n \in \Sigma^\omega$ we define $zip(t_1, \cdots, t_n) \in (\Sigma^n)^\omega$ as the word obtained by combining the traces pointwise, i.e., $zip(t_1, \cdots, t_n)(i) := (t_1(i), \cdots, t_n(i))$. Our algorithm now progresses in a bottom-up manner. Assume that some subformula φ occurs under quantifiers binding path variables $\pi_1, \cdots, \pi_n$. We say that an automaton $\mathcal{A}$ over S^n is $\mathcal{G}$-equivalent to φ, if for any paths $t_1, \cdots, t_n$ it holds that $[\pi_i \mapsto t_i]_{i=1}^n \models_{\mathcal{G}} \varphi$ if any only if $zip(t_1, \cdots, t_n) \in \mathcal{L}(\mathcal{A})$. $\mathcal{G}$-equivalence thus means that an automaton accepts a zipping of traces exactly if the trace assignment constructed from those traces satisfies the formula. By induction on the structure of the formula, we construct an automaton that is $\mathcal{G}$-equivalent to each sub formula.

For the standard boolean combinators and `LTL` temporal operators our construction follows the typical translation from `LTL` to alternating automata [34, 43] given in Fig. 2. The interesting case is now the elimination of a strategy quantifier of the form $\varphi = \langle\!\langle A \rangle\!\rangle \pi.\psi$. Given an inductively constructed APA $\mathcal{A}_\psi$ over $\Sigma_\psi = S^{n+1}$. We aim for an automaton $\mathcal{A}_\varphi$ over $\Sigma_\varphi = S^n$. The automata should accept all traces t over S^n such that there exist a strategy for agents in A such that all traces compatible with this strategy t' when added to t (the trace $t \times t' \in (S^{n+1})^\omega$) is accepted by $\mathcal{A}_\psi$. Let $\mathcal{G} = (S, s_0, \Xi, \mathcal{M}, \delta, d, \mathbf{AP}, L)$ be the given MSCGS. We distinguish between the cases where $A = \Xi$ (i.e., existential quantification) and $A \neq \Xi$.

Existential Quantification. We first consider the case where $A = \Xi$, i.e., $\varphi = \exists \pi.\psi$. Model checking can be done similar to [20]. Let $\mathcal{A}_\psi = (Q, q_0, \Sigma_\psi, \lambda : Q \times \Sigma_\psi \to 2^Q, c)$ be the inductively constructed automaton, translated into a non-deterministic automaton of exponential size via Theorem 1. We then construct $\mathcal{A}_\varphi := (S \times Q \cup \{q_{init}\}, q_{init}, \Sigma_\varphi, \rho, c')$ where $c'(s, q) = c(q)$ ($c'(q_{init})$ can be chosen arbitrarily) and ρ is defined via

$$\rho(q_{init}, [s_1, \cdots, s_n]) = \{(s', q') \mid q' \in \lambda(q, [s_1, \cdots, s_n, s_n^\circ]) \land \exists \sigma : \Xi \to \mathcal{M}. \delta(s_n^\circ, \sigma) = s'\}$$
$$\rho((s, q), [s_1, \cdots, s_n]) = \{(s', q') \mid q' \in \lambda(q, [s_1, \cdots, s_n, s]) \land \exists \sigma : \Xi \to \mathcal{M}. \delta(s, \sigma) = s'\}$$

where we define $s_n^\circ = s_n$ if $n \geq 1$ and $s_n^\circ = s_0$ (the initial state) otherwise. Note that $\mathcal{A}_\varphi$ is again a non-deterministic automaton. Every accepting run of $\mathcal{A}_\varphi$ on $zip(t_1, \cdots, t_n)$ now corresponds to a path t in $\mathcal{G}$ such that $\mathcal{A}_\psi$ accepts $zip(t_1, \cdots, t_n, t)$.

(Complex) Strategic Quantification. We now consider the case where $A \neq \Xi$. Our automaton must encode the strategic behaviour of the agents. We achieve this, by encoding the strategic play in the game structure within the transition function of an automaton. Let $\mathcal{A}_\psi^{det} = (Q, q_0, \Sigma_\psi, \lambda : Q \times \Sigma_\psi \to Q, c)$ be a *deterministic* automaton obtained from the inductively constructed $\mathcal{A}_\psi$ via Theorem 1. Note that $\mathcal{A}_\psi^{det}$ is, in the worst case, of double exponential size (in the size of $\mathcal{A}_\psi$). To encode the strategic behaviour in $\mathcal{G}$ we use the alternation available in an automaton by disjunctively choosing moves for controlled players in A, followed by a conjunctive treatment of all adversarial player. The stages of a game, naturally correspond to the order of the move selection. Define the set $A_i := A \cap d^{-1}(i)$ and $\overline{A}_i := (\Xi \setminus A) \cap d^{-1}(i)$ and let m be the maximal element in the codomain of d. The choice of each agent in A followed by those not in A can be encoded into a boolean formula. We define $\mathcal{A}_\varphi := (S \times Q \cup \{q_{init}\}, q_{init}, \Sigma_\varphi, \rho, c')$ where ρ is defined by

$$\begin{aligned}&\rho(q_{init},[s_1,\cdots,s_n])\\&\quad= \bigvee_{\sigma_1:A_1\to\mathcal{M}}\ \bigwedge_{\sigma'_1:\overline{A}_1\to\mathcal{M}} \cdots \bigvee_{\sigma_m:A_m\to\mathcal{M}}\ \bigwedge_{\sigma'_m:\overline{A}_m\to\mathcal{M}} \big(\delta(s_n^\circ, \sum_{i=1}^{m}(\sigma_i+\sigma'_i)), \lambda(q_0,[s_1,\cdots,s_n,s_n^\circ])\big)\end{aligned}$$

$$\begin{aligned}&\rho((s,q),[s_1,\cdots,s_n])\\&\quad= \bigvee_{\sigma_1:A_1\to\mathcal{M}}\ \bigwedge_{\sigma'_1:\overline{A}_1\to\mathcal{M}} \cdots \bigvee_{\sigma_m:A_m\to\mathcal{M}}\ \bigwedge_{\sigma'_m:\overline{A}_m\to\mathcal{M}} \big(\delta(s, \sum_{i=1}^{m}(\sigma_i+\sigma'_i)), \lambda(q,[s_1,\cdots,s_n,s])\big)\end{aligned}$$

and $c'(s,q) = c(q)$ (we can again define $c'(q_{init})$ arbitrarily). In case $n = 0$, we again define s_n° as the initial state s_0, otherwise $s_n^\circ = s_n$. Note that in the case where the MSCGS is a CGS, i.e., $d = \mathbf{0}$ the transition function has the form $\vee\wedge$, where the choices in A are considered disjunctively and the choices by all other agents conjunctively. Our construction can be extended to handle the self composition $[\langle\langle A_1\rangle\rangle\pi_1.\ \cdots\ \langle\langle A_k\rangle\rangle\pi_k]$ (see [6] for details).

Negated Quantification. We extend our construction to handle negation outside of quantifiers, i.e., a formula $\varphi = \neg\langle\langle A\rangle\rangle\pi.\psi$ via $\mathcal{A}_\varphi := \overline{\mathcal{A}_{\langle\langle A\rangle\rangle\pi.\psi}}$ by using Theorem 2.

▶ **Proposition 8.** *$\mathcal{A}_\varphi$ is $\mathcal{G}$-equivalent to φ.*

By following our inductive construction, we obtain an automaton over the singleton alphabet (empty state sequences) that is non-empty iff the model satisfies the formula. Emptiness of an alternating parity automaton can then be checked in polynomial size (assuming a fixed number of colours) [28].

We can observe a gap in complexity of algorithm between simple and complex quantification. The former case requires a translation of an alternating automaton to a non-deterministic one whereas the latter requires a full determinisation. To capture the complexity of our algorithm we define $\mathcal{T}_c(k,n)$ as a tower of k exponents in n, i.e., $\mathcal{T}_c(0,n) = n^c$ and $\mathcal{T}_c(k+1,n) = c^{\mathcal{T}_c(k,n)}$. For $k \geq 0$ we define k-EXPSPACE as the class of languages recognised by a deterministic Turing machine ([41]) with space $\mathcal{T}_c(k,\mathcal{O}(n))$ for some c (and similarly for time). We define (-1)-EXPSPACE as NLOGSPACE. Note that 0-EXPSPACE = PSPACE.

▶ **Theorem 9.** *Model checking of a HyperATL* formula with k complex and l simple quantifiers is in $(2k+l)$-EXPTIME. If $l \geq 1$ and the formula is linear, it is also in $(2k+l-1)$-EXPSPACE (both in size of the formula).*

The fact that we can derive a better upper bound when $l > 0$ follows from the fact that we can determine the emptiness of a non-deterministic automaton in NLOGSPACE [44] and for an alternating automaton only in polynomial time (for a fixed number of colours) [28]. Note that for the syntactic fragment of HyperCTL* our algorithm matches the algorithm in [20].

6 Lower Bounds for Model Checking

Theorem 9 gives us an upper bound on the model checking problem for HyperATL*. We can show the following lower bound

▶ **Theorem 10.** *Model checking of a linear HyperATL* formula with k complex and l simple quantifiers is $(2k+l-1)$-EXPSPACE-hard in the size of the formula, provided $l \geq 1$.*

The proof of Theorem 10 proceeds by encoding space-bounded Turing machines into HyperATL*. We show that (complex) strategic quantification can be used to encode an incremental counter that grows by *two exponents* with each quantifier, opposed to the increment by a single exponent for simple quantification [38]. This is possible, as we can design a formula that requires the first player to enumerate a counter, while the second play can challenge its correctness. The only winning strategy for the former player is then to output a correct counter that holds up against all scrutiny by the latter player. As the construction of the counter is rather complex, we refer the interested reader to a detailed proof in the full version [6].

Note that Theorem 10 is conditioned on $l \geq 1$. This gives an interesting complexity landscape: In cases where $l \geq 1$, model checking is $(2k+l-1)$-EXPSPACE-complete (irrespective of k). If $l = 0$ we get an upper bound of $2k$-EXPTIME (Theorem 9). In the special case where $l = 0$ and $k = 1$ we get a matching lower bound from the ATL* model-checking problem [1] (subsuming LTL realizability [36, 37]) and thus 2-EXPTIME-completeness. If $k > 1$ the best lower bound is $(2k-2)$-EXPSPACE. The exact complexity for the case where $k > 1$ and $l = 0$ is thus still open.

7 HyperATL* vs. asynchronous HyperLTL

We have seen that our strategic logic can naturally express asynchronous hyperproperties. In this section, we compare our logic to AHLTL [4], a recent extension of HyperLTL specifically designed to express such asynchronous properties. AHLTL is centred around the idea of a trajectory, which, informally speaking, is the stuttering of traces in a system. In AHLTL an initial trace quantifier prefix is followed by a quantification over such a trajectory. For example, a formula of the form $\forall \pi_1. \cdots \forall \pi_n.\mathbf{E}.\varphi$ means that for all paths $\pi_1, \cdots, \pi_n$ in the system there exists some stuttering of the paths, such that φ is satisfied. AHLTL follows a purely trace-based approach where the stuttering is fixed, knowing the full paths $\pi_1, \cdots, \pi_n$. In comparison, in our logic a *strategy* must decide if to stutter based on finite a prefix in the system. Model checking AHLTL is, in general, undecidable [4]. The largest known fragment for which an algorithm is known are formulas of the form $\forall \pi_1. \cdots \forall \pi_n.\mathbf{E}.\varphi$ where φ is an *admissible formula* [4] which is a conjunction of formulas of the form $\Box \bigwedge_{a \in P} a_{\pi_i} \leftrightarrow a_{\pi_j}$ (where P is a set of atomic propositions) and stutter-invariant formulas over a single path variable. We can show the following (where $\mathcal{G}_{stut}$ is the stutter transformation from Definition 7):

▶ **Theorem 11.** *For any Kripke structure $\mathcal{G}$ and AHLTL formula of the form $\forall \pi_1. \cdots \forall \pi_n.\mathbf{E}.\varphi$ it holds that if $\mathcal{G}_{stut} \models [\langle\langle \{sched\} \rangle\rangle \pi_1. \cdots \langle\langle \{sched\} \rangle\rangle \pi_n]\ \varphi \wedge \bigwedge_{i \in \{1, \cdots, n\}} fair_{\pi_i}$ (1) then $\mathcal{G} \models_{AHLTL} \forall \pi_1. \cdots \forall \pi_n.\mathbf{E}.\varphi$ (2). If φ is an admissible formula, (1) and (2) are equivalent.*

Theorem 11 gives us a sound approximation of the (undecidable) AHLTL model checking. Furthermore, for admissible formulas, AHLTL can be truthfully expressed in our logic. As shown in [4], many interesting properties can be expressed using an admissible formula and can thus be (truthfully) checked in our logic. Our framework can therefore express many interesting properties, is fully decidable, and also subsumes the largest known decidable fragment of AHLTL.

8 Experimental Evaluation

While MC for the full logic is very expensive (Theorem 10) and likely not viable in practice, formulas of the form $[\langle\langle A_1 \rangle\rangle \pi_1. \ \cdots \langle\langle A_n \rangle\rangle \pi_n.]\varphi$ where φ is quantifier free, can be checked efficiently via a reduction to a parity game (see the full version [6] for details). Note that

Table 1 Validity of various `HyperATL`* formulas on small benchmark programs. A ✓(resp. ✗) means that the formula is satisfied (resp. not satisfied). The time consumption is given in milliseconds.

	(OD)	(NI)	(simSec)	(sGNI)
`P1`	✓(15)	✓(16)	✓(16)	✓(46)
`P2`	✗(112)	✓(80)	✓(83)	✓(432)
`P3`	✗(70)	✗(44)	✓(54)	✓(112)
`P4`	✗(73)	✗(64)	✗(70)	✓(191)

(a) Examples for Information-Flow Policies.

	(OD)	($\mathbf{OD}_{asynch}$)	($\mathbf{NI}_{asynch}$)
$\texttt{Q1}_i$	✗(112)	✓(788)	✓(812)
$\texttt{Q1}_{ii}$	✗(281)	✓(3372)	✓(3516)
$\texttt{Q1}_{iii}$	✗(1680)	✓(20756)	✓(24078)
`Q2`	✗(985)	✗(18141)	✓(6333)

(b) Examples for Asynchronous Hyperproperties.

all alternation-free `HyperLTL` formulas, the reduction from the MC approach from [12] and the reduction in Theorem 11 fall in this fragment. We implemented a prototype model checker for this fragment to demonstrate different security notions (both synchronous and asynchronous) on small example programs. Our tool uses `rabinizer 4` [27] to convert a `LTL` formula into a deterministic automaton and `pgsolver` [22] to solve parity games. Our tool is publicly available (see the full version [6]).

Information-Flow Policies. We have created a small benchmark of simple programs that distinguish different information-flow policies. We checked the following properties: **(OD)** is the standard (alternation-free) formula of observational determinism, **(NI)** is a simple formulation of non-interference due to [23], **(simSec)** is simulation security [39] as expressed in Sec. 4.1. Finally, **(sGNI)** is the simple game based definition of GNI resolved on the parallel-composition (as used in [12]). We designed small example programs that demonstrate the difference between security guarantees and present the results in Table 1a. Note that the model checking algorithm for $\forall^*\exists^*$ formulas from [12] is subsumed by our approach. As we reduce the search for a strategy for the existential player to a parity game opposed to a SMT constraint, we can handle much bigger systems.

Asynchronous Hyperproperties. To showcase the expressiveness of our framework to handle asynchronous properties, we implemented the stuttering transformation from Definition 7. We evaluated our tool by checking example programs both on synchronous observational-determinism **(OD)** and asynchronous versions of OD ($\mathbf{OD}_{asynch}$) and non-interference $(\mathbf{NI})_{asynch}$. Note that while ($\mathbf{OD}_{asynch}$) can also express in the decidable fragment of `AHLTL`, ($\mathbf{NI}_{asynch}$) is not an admissible formula (and can not be handled in [4]). As non-interference only requires the outputs to align provided the inputs do, one needs to take care that the asynchronous scheduler does not “cheat“ by deliberately missaligning inputs and thereby invalidating the premise of this implication. Our results are given in Table 1b. To demonstrate the state-explosion problem we tested the same program (`Q1`) with different bit-widths (programs $\texttt{Q1}_i$, $\texttt{Q1}_{ii}$, $\texttt{Q1}_{iii}$), causing an artificial blow-up in the number of states.

9 Related Work

There has been a lot of recent interest in logics for hyperproperties. Most logics are obtained by extending standard temporal or first-order/second-order logics with either path quantification or by a special equal-level predicate [21]. See [11] for an overview. To the best of our knowledge, none of these logics can express strategic hyperproperties.

Alternating-time Temporal Epistemic Logic. The relationship between epistemic logics and hyperlogic is interesting, as both reason about the flow of information in a system. As shown in [7], HyperLTL and $\mathtt{LTL}_{\mathcal{K}}$ (LTL extended with a knowledge operator [15]) have incomparable expressiveness. In HyperQPTL, which extends HyperLTL with propositional quantification [18], the knowledge operator can be encoded by explicitly marking the knowledge positions via propositional quantification [38, §7]. Alternating-time temporal logic has also been extended with knowledge operators [42]. The resulting logic, ATEL, can express properties of the form "if ξ knows ϕ, then she can enforce ψ via a strategy." The natural extension of the logic in [42], which allows for arbitrary nesting of quantification and operators (i.e., an extension of ATL* instead of ATL) is incomparable to HyperATL*.

Model Checking. Decidable model checking is a crucial prerequisite for the effective use of a logic. Many of the existing (synchronous) hyperlogics admit decidable model checking, although mostly with non-elementary complexity (see [17] for an overview). For alternating-time temporal logic (in the non-hyper realm), model checking is efficient (especially when one prohibits arbitrary nesting of temporal operators and quantifiers as in ATL) [1, 2]. If one allows operators and quantifiers to be nested arbitrarily (ATL*), model checking subsumes LTL satisfiability and realizability. This causes a jump in the model checking complexity to 2-EXPTIME-completeness. As our lower bound demonstrates, the combination of strategic quantification and hyperproperties results in a logic that is algorithmically harder (for model checking) than non-strategic hyperproperties (as HyperLTL) or non-hyper strategic logics (as ATL*). The fragment of HyperATL* implemented in our prototype model checker subsumes alteration-free HyperLTL (see MCHyper [20]), model checking via explicit strategies [12] and the (known) decidable fragment of AHLTL [4].

Asynchronous Hyperproperties. Extending hyperlogics to express asynchronous properties has only recently started to gain momentum [25, 4, 8]. In [4] they extend HyperLTL with explicit trajectory quantification. [25] introduced a variant of the polyadic μ-calculus, H_μ, able to express hyperproperties. In [8] they extended HyperLTL with new modalities that remove redundant (for example stuttering) parts of a trace. Model checking is undecidable for all three logics. The (known) decidable fragment of [4] can be encoded into HyperATL*. The only known decidable classes for H_μ [25] and $\mathtt{HyperLTL}_S$ [8] are obtained by bounding the asynchronous offset by a constant k, i.e., asynchronous execution may not run apart ("diverge") for more than k steps. For actual software, this is a major restriction.

10 Conclusion

We have introduced HyperATL*, a temporal logic for strategic hyperproperties. Besides the obvious benefits of simultaneously reasoning about strategic choice and information flow, HyperATL*provides a natural formalism to express *asynchronous* hyperproperties, which has been a major challenge for previous hyperlogics. Despite the added expressiveness, HyperATL* model checking remains decidable, with comparable cost to logics for synchronous hyperproperties (cf. Theorem 9). HyperATL* is the first logic for asynchronous hyperproperties where model checking is decidable for the entire logic. Its expressiveness and decidability, as well as the availability of practical model checking algorithms, make it a very promising choice for model checking tools for hyperproperties.

References

1 Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002. doi:10.1145/585265.585270.

2 Rajeev Alur, Thomas A. Henzinger, Freddy Y. C. Mang, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. MOCHA: modularity in model checking. In *Computer Aided Verification, 10th International Conference, CAV '98, Vancouver, BC, Canada, June 28 - July 2, 1998, Proceedings*, volume 1427 of *Lecture Notes in Computer Science*, pages 521–525. Springer, 1998. doi:10.1007/BFb0028774.

3 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

4 Jan Baumeister, Norine Coenen, Borzoo Bonakdarpour, Bernd Finkbeiner, and César Sánchez. A temporal logic for asynchronous hyperproperties. In *Computer Aided Verification - 33nd International Conference, CAV 2021, Los Angeles, CA, USA, July 18-24, 2021*, Lecture Notes in Computer Science. Springer, 2021.

5 Raphaël Berthon, Bastien Maubert, and Aniello Murano. Decidability results for atl* with imperfect information and perfect recall. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, pages 1250–1258. ACM, 2017. URL: http://dl.acm.org/citation.cfm?id=3091299.

6 Raven Beutner and Bernd Finkbeiner. A temporal logic for strategic hyperproperties. *CoRR*, abs/2107.02509, 2021. arXiv:2107.02509.

7 Laura Bozzelli, Bastien Maubert, and Sophie Pinchinat. Unifying hyper and epistemic temporal logics. In *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9034 of *Lecture Notes in Computer Science*, pages 167–182. Springer, 2015. doi:10.1007/978-3-662-46678-0_11.

8 Laura Bozzelli, Adriano Peron, and César Sánchez. Asynchronous extensions of hyperltl. In *36nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021.* ACM, 2021.

9 Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and César Sánchez. Temporal logics for hyperproperties. In *Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, volume 8414 of *Lecture Notes in Computer Science*, pages 265–284. Springer, 2014. doi:10.1007/978-3-642-54792-8_15.

10 Michael R. Clarkson and Fred B. Schneider. Hyperproperties. *J. Comput. Secur.*, 18(6):1157–1210, 2010. doi:10.3233/JCS-2009-0393.

11 Norine Coenen, Bernd Finkbeiner, Christopher Hahn, and Jana Hofmann. The hierarchy of hyperlogics. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785713.

12 Norine Coenen, Bernd Finkbeiner, César Sánchez, and Leander Tentrup. Verifying hyperliveness. In *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 121–139. Springer, 2019. doi:10.1007/978-3-030-25540-4_7.

13 Doron Drusinsky and David Harel. On the power of bounded concurrency I: finite automata. *J. ACM*, 41(3):517–539, 1994. doi:10.1145/176584.176587.

14 E. Allen Emerson and Joseph Y. Halpern. "Sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):151–178, 1986. doi:10.1145/4904.4999.

15 Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning About Knowledge.* MIT Press, 1995. doi:10.7551/mitpress/5803.001.0001.

16 Bernd Finkbeiner. Temporal hyperproperties. *Bull. EATCS*, 123, 2017. URL: http://eatcs.org/beatcs/index.php/beatcs/article/view/514.

17 Bernd Finkbeiner. Model checking algorithms for hyperproperties (invited paper). In *Verification, Model Checking, and Abstract Interpretation - 22nd International Conference, VMCAI 2021, Copenhagen, Denmark, January 17-19, 2021, Proceedings*, volume 12597 of *Lecture Notes in Computer Science*, pages 3–16. Springer, 2021. doi:10.1007/978-3-030-67067-2_1.

18 Bernd Finkbeiner, Christopher Hahn, Jana Hofmann, and Leander Tentrup. Realizing omega-regular hyperproperties. In *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 40–63. Springer, 2020. doi:10.1007/978-3-030-53291-8_4.

19 Bernd Finkbeiner, Christopher Hahn, Philip Lukert, Marvin Stenger, and Leander Tentrup. Synthesizing reactive systems from hyperproperties. In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pages 289–306. Springer, 2018. doi:10.1007/978-3-319-96145-3_16.

20 Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking hyperltl and hyperctl*. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 30–48. Springer, 2015. doi:10.1007/978-3-319-21690-4_3.

21 Bernd Finkbeiner and Martin Zimmermann. The first-order logic of hyperproperties. In *34th Symposium on Theoretical Aspects of Computer Science, STACS 2017, March 8-11, 2017, Hannover, Germany*, volume 66 of *LIPIcs*, pages 30:1–30:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.STACS.2017.30.

22 Oliver Friedmann and Martin Lange. Solving parity games in practice. In *Automated Technology for Verification and Analysis, 7th International Symposium, ATVA 2009, Macao, China, October 14-16, 2009. Proceedings*, volume 5799 of *Lecture Notes in Computer Science*, pages 182–196. Springer, 2009. doi:10.1007/978-3-642-04761-9_15.

23 Joseph A. Goguen and José Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy, Oakland, CA, USA, April 26-28, 1982*, pages 11–20. IEEE Computer Society, 1982. doi:10.1109/SP.1982.10014.

24 Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Propositional dynamic logic for hyperproperties. In *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 50:1–50:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.50.

25 Jens Oliver Gutsfeld, Markus Müller-Olm, and Christoph Ohrem. Automata and fixpoints for asynchronous hyperproperties. *Proc. ACM Program. Lang.*, 5(POPL):1–29, 2021. doi:10.1145/3434319.

26 Marieke Huisman, Pratik Worah, and Kim Sunesen. A temporal logic characterisation of observational determinism. In *19th IEEE Computer Security Foundations Workshop, (CSFW-19 2006), 5-7 July 2006, Venice, Italy*, page 3. IEEE Computer Society, 2006. doi:10.1109/CSFW.2006.6.

27 Jan Kretínský, Tobias Meggendorfer, Salomon Sickert, and Christopher Ziegler. Rabinizer 4: From LTL to your favourite deterministic automaton. In *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part I*, volume 10981 of *Lecture Notes in Computer Science*, pages 567–577. Springer, 2018. doi:10.1007/978-3-319-96145-3_30.

28 Orna Kupferman and Moshe Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 224–233. ACM, 1998. doi:10.1145/276698.276748.

29 Orna Kupferman and Moshe Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001. doi:10.1145/377978.377993.

30 Heiko Mantel and Henning Sudbrock. Flexible scheduler-independent security. In *Computer Security - ESORICS 2010, 15th European Symposium on Research in Computer Security,*

Athens, Greece, September 20-22, 2010. Proceedings, volume 6345 of *Lecture Notes in Computer Science*, pages 116–133. Springer, 2010. doi:10.1007/978-3-642-15497-3_8.

31 Daryl McCullough. Noninterference and the composability of security properties. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 18-21, 1988*, pages 177–186. IEEE Computer Society, 1988. doi:10.1109/SECPRI.1988.8110.

32 Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980. doi:10.1007/3-540-10235-3.

33 Satoru Miyano and Takeshi Hayashi. Alternating finite automata on omega-words. *Theor. Comput. Sci.*, 32:321–330, 1984. doi:10.1016/0304-3975(84)90049-5.

34 David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *Proceedings of the Third Annual Symposium on Logic in Computer Science (LICS '88), Edinburgh, Scotland, UK, July 5-8, 1988*, pages 422–427. IEEE Computer Society, 1988. doi:10.1109/LICS.1988.5139.

35 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 46–57. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

36 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190. ACM Press, 1989. doi:10.1145/75277.75293.

37 Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In *Automata, Languages and Programming, 16th International Colloquium, ICALP89, Stresa, Italy, July 11-15, 1989, Proceedings*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer, 1989. doi:10.1007/BFb0035790.

38 Markus N. Rabe. *A temporal logic approach to information-flow control.* PhD thesis, Saarland University, 2016. URL: http://scidok.sulb.uni-saarland.de/volltexte/2016/6387/.

39 Andrei Sabelfeld. Confidentiality for multithreaded programs via bisimulation. In *Perspectives of Systems Informatics, 5th International Andrei Ershov Memorial Conference, PSI 2003, Akademgorodok, Novosibirsk, Russia, July 9-12, 2003, Revised Papers*, volume 2890 of *Lecture Notes in Computer Science*, pages 260–274. Springer, 2003. doi:10.1007/978-3-540-39866-0_27.

40 Andrei Sabelfeld and David Sands. Probabilistic noninterference for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop, CSFW '00, Cambridge, England, UK, July 3-5, 2000*, pages 200–214. IEEE Computer Society, 2000. doi:10.1109/CSFW.2000.856937.

41 Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970. doi:10.1016/S0022-0000(70)80006-X.

42 Wiebe van der Hoek and Michael J. Wooldridge. Cooperation, knowledge, and time: Alternating-time temporal epistemic logic and its applications. *Stud Logica*, 75(1):125–157, 2003. doi:10.1023/A:1026185103185.

43 Moshe Y. Vardi. Alternating automata and program verification. In *Computer Science Today: Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science*, pages 471–485. Springer, 1995. doi:10.1007/BFb0015261.

44 Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Inf. Comput.*, 115(1):1–37, 1994. doi:10.1006/inco.1994.1092.

45 J. Todd Wittbold and Dale M. Johnson. Information flow in nondeterministic systems. In *Proceedings of the 1990 IEEE Symposium on Security and Privacy, Oakland, California, USA, May 7-9, 1990*, pages 144–161. IEEE Computer Society, 1990. doi:10.1109/RISP.1990.63846.

Time Flies When Looking out of the Window: Timed Games with Window Parity Objectives

James C. A. Main
UMONS – Université de Mons, Belgium

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

Jeremy Sproston
University of Torino, Italy

Abstract

The *window mechanism* was introduced by Chatterjee et al. to reinforce mean-payoff and total-payoff objectives with time bounds in two-player turn-based games on graphs [17]. It has since proved useful in a variety of settings, including parity objectives in games [14] and both mean-payoff and parity objectives in Markov decision processes [12].

We study *window parity* objectives in timed automata and timed games: given a bound on the window size, a path satisfies such an objective if, in all states along the path, we see a sufficiently small window in which the smallest priority is even. We show that checking that all time-divergent paths of a timed automaton satisfy such a window parity objective can be done in polynomial space, and that the corresponding timed games can be solved in exponential time. This matches the complexity class of timed parity games, while adding the ability to reason about time bounds. We also consider multi-dimensional objectives and show that the complexity class does not increase. To the best of our knowledge, this is the first study of the window mechanism in a real-time setting.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases Window objectives, timed automata, timed games, parity games

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.25

Related Version *Full Version*: `https://arxiv.org/abs/2105.06686` [23]

Funding Research supported by F.R.S.-FNRS under Grant n° F.4520.18 (ManySynth).
Mickael Randour: F.R.S.-FNRS Research Associate.

1 Introduction

Timed automata and games. *Timed automata* [2] are extensions of finite automata with real-valued variables called *clocks*. Clocks increase at the same rate and measure the elapse of time between actions. Transitions are constrained by the values of clocks, and clocks can be reset on transitions. Timed automata are used to model real-time systems [4]. Not all paths of timed automata are meaningful; infinite paths that take a finite amount of time, called *time-convergent* paths, are often disregarded when checking properties of timed automata. Timed automata induce uncountable transition systems. However, many properties can be checked using the region abstraction, which is a finite quotient of the transition system.

Timed automaton games [24], or simply *timed games*, are games played on timed automata: one player represents the system and the other its environment. Players play an infinite amount of rounds: for each round, both players simultaneously present a delay and an action, and the play proceeds according to the fastest move (note that we use paths for automata and plays for games to refer to sequences of consecutive states and transitions). When defining winning conditions for players, convergent plays must be taken in account; we must not allow a player to achieve its objective by forcing convergence but cannot either require a player to

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 25; pp. 25:1–25:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

force divergence (as it also depends on its opponent). Given an objective as a set of plays, following [20], we declare a play winning for a player if either it is *time-divergent and belongs to the objective*, or it is *time-convergent and the player is not responsible for convergence.*

Parity conditions. The class of ω-regular specifications is widely used (e.g., it can express liveness and safety), and parity conditions are a canonical way of representing them. In (timed) parity games, locations are labeled with a non-negative integer priority and the parity objective is to ensure the smallest priority occurring infinitely often along the path/play is even. Timed games with ω-regular objectives given as parity automata are shown to be solvable in [20]. Furthermore, a reduction from timed parity games to classical turn-based parity games on a graph is established in [19].

Real-timed windows. The parity objective can be reformulated: for all odd priorities seen infinitely often, a smaller even priority must be seen infinitely often. One can see the odd priority as a request and the even one as a response. The parity objective does not specify any *timing constraints* between requests and responses. In applications however, this may not be sufficient: for example, a server should respond to requests in a timely manner.

We revisit the window mechanism introduced by Chatterjee et al. for mean-payoff and total-payoff games [17] and later applied to parity games [14] and to parity and mean-payoff objectives in Markov decision processes [12]: we provide *the first* (to the best of our knowledge) *study of window objectives in the real-time setting.* More precisely, we lift the (resp. direct) fixed window parity objective of [14] to its real-time counterpart, the *(resp. direct) timed window parity objective*, and study it in timed automata and games.

Intuitively, given a non-negative integer bound λ on the window size, the *direct* timed window parity objective requires that *at all* times along a path/play, we see a window of size at most λ such that the smallest priority in this window is even. While time was counted as steps in prior works (all in a discrete setting), *we naturally measure window size using delays* between configurations in real-time models. The *(non-direct)* timed window parity objective is simply a prefix-independent version of the direct one, thus more closely matching the spirit of classical parity: it asks that some suffix satisfies the direct objective.

Contributions. We extend window parity objectives to a dense-time setting, and study both *verification* of timed automata and *realizability* in timed games. We consider adaptations of the *fixed window parity objectives* of [14], where the window size is given as a parameter. We establish that (a) verifying that all time-divergent paths of a timed automaton satisfy a timed window parity specification is PSPACE-complete; and that (b) checking the existence of a winning strategy for a window parity objective in timed games is EXPTIME-complete. These results (Thm. 8) hold for both the direct and prefix-independent variants, and they extend to multi-dimensional objectives, i.e., conjunctions of window parity.

All algorithms are based on a reduction to an *expanded timed automaton* (Def. 4). We establish that, similarly to the discrete case, it suffices to keep track of one window at a time (or one per objective in the multi-dimensional case) instead of all currently open windows, thanks to the so-called *inductive property of windows* (Lem. 3). A window can be summarized using its smallest priority and its current size: we encode the priorities in a window by extending locations with priorities and using an additional clock to measure the window's size. The (resp. direct) timed window parity objective translates to a co-Büchi (resp. safety) objective on the expanded automaton. Locations to avoid for the co-Büchi (resp. safety) objective indicate a window exceeding the supplied bound without the smallest priority of the window being even – a *bad window*. To check that all time-divergent paths of the expanded automaton satisfy the safety (resp. co-Büchi) objective, we check for the existence of a time-divergent path visiting (resp. infinitely often) an unsafe location using the PSPACE algorithm of [2]. To solve the similarly-constructed expanded game, we use the EXPTIME algorithm of [20].

Lower bounds are established by encoding safety objectives on timed automata as (resp. direct) timed window parity objectives. Checking safety properties over time-divergent paths in timed automata is PSPACE-complete [2] and solving safety timed games is EXPTIME-complete [21].

Comparison. Window variants constitute *conservative approximations* of classical objectives (e.g., [17, 14, 12]), strengthening them by enforcing timing constraints. Complexity-wise, the situation is varied. In one-dimension turn-based games on graphs, window variants [17, 14] provide *polynomial-time alternatives* to the classical objectives, bypassing long-standing complexity barriers. However, in multi-dimension games, their complexity becomes worse than for the original objectives: in particular, fixed window parity games are EXPTIME-complete for multiple dimensions [14]. We show that timed games with multi-dimensional window parity objectives are in the same complexity class as untimed ones, i.e., dense time comes for free.

For classical parity objectives, timed games can be solved in exponential time [19, 20]. The approach of [19] is as follows: from a timed parity game, one builds a corresponding turn-based parity game on a graph, the construction being polynomial in the number of priorities and the size of the region abstraction. We recall that despite recent progress on quasi-polynomial-time algorithms (starting with [16]), no polynomial-time algorithm is known for parity games; the blow-up comes from the number of priorities. Overall, *the two sources of blow-up* – region abstraction and number of priorities – *combine in a single-exponential solution* for timed parity games. We establish that (multi-dimensional) window parity games can be solved in time polynomial in the size of the region abstraction, the number of priorities and the window size, and exponential in the number of dimensions. Thus *even for conjunctions of objectives, we match the complexity class of single parity objectives of timed games, while avoiding the blow-up related to the number of priorities and enforcing time bounds between odd priorities and smaller even priorities via the window mechanism.*

Outline. Due to space constraints, we only provide an overview of our work. All technical details, intermediary results and proofs can be found in the full version of this paper [23]. This work is organized as follows. Sect. 2 summarizes all prerequisite notions and vocabulary. In Sect. 3, we introduce the timed window parity objective, compare it to the classical parity objective, and establish some useful properties. Our reduction from window parity objectives to safety or co-Büchi objectives is presented in Sect. 4. Finally, Sect. 5 presents complexity results.

Related work. In addition to the aforementioned foundational works, the window mechanism has seen diverse extensions and applications: e.g., [5, 3, 11, 15, 22, 26, 8]. Window parity games are strongly linked to the concept of *finitary ω-regular games*: see, e.g., [18], or [14] for a complete list of references. The window mechanism can be used to ensure a certain form of (local) guarantee over paths: different techniques have been considered in stochastic models, notably variance-based [10] or worst-case-based [13, 7] methods. Finally, let us recall that game models provide a useful framework for controller synthesis [25], and that timed automata have been extended in a number of directions (see, e.g., [9] and references therein): applications of the window mechanism in such richer models could be of interest.

2 Preliminaries

Timed automata. A clock variable, or *clock*, is a real-valued variable. Let C be a set of clocks. A *clock constraint* over C is a conjunction of formulae of the form $x \sim c$ with $x \in C$, $c \in \mathbb{N}$, and $\sim \in \{\leq, \geq, >, <\}$. We write $x = c$ as shorthand for the clock constraint $x \geq c \wedge x \leq c$. Let $\Phi(C)$ denote the set of clock constraints over C.

Let $\mathbb{R}_{\geq 0}$ denote the set of non-negative real numbers. We refer to functions $\nu \in \mathbb{R}^C_{\geq 0}$ as *clock valuations* over C. A clock valuation ν over C satisfies a clock constraint of the form $x \sim c$ if $\nu(x) \sim c$ and a conjunction $g \wedge h$ if it satisfies both g and h. For a clock constraint g and clock valuation ν, we write $\nu \models g$ if ν satisfies g.

For a clock valuation ν and $d \geq 0$, we let $\nu + d$ be the valuation defined by $(\nu + d)(x) = \nu(x) + d$ for all $x \in C$. For any valuation ν and $D \subseteq C$, we define $\mathsf{reset}_D(\nu)$ to be the valuation agreeing with ν for clocks in $C \setminus D$ and that assigns 0 to clocks in D. We denote by $\mathbf{0}^C$ the zero valuation, assigning 0 to all clocks in C.

A *timed automaton* (TA) is a tuple $(L, \ell_{\mathsf{init}}, C, \Sigma, I, E)$ where L is a finite set of *locations*, $\ell_{\mathsf{init}} \in L$ is an initial location, C a finite set of *clocks* containing a special clock γ which keeps track of the total time elapsed, Σ a finite set of actions, $I \colon L \to \Phi(C)$ an *invariant* assignment function and $E \subseteq L \times \Phi(C) \times \Sigma \times 2^{C \setminus \{\gamma\}} \times L$ an edge relation. We only consider deterministic timed automata, i.e., we assume that in any location ℓ, there are no two outgoing edges $(\ell, g_1, a, D_1, \ell_1)$ and $(\ell, g_2, a, D_2, \ell_2)$ sharing the same action such that the conjunction $g_1 \wedge g_2$ is satisfiable. For an edge (ℓ, g, a, D, ℓ'), the clock constraint g is called the *guard* of the edge.

A TA $\mathcal{A} = (L, \ell_{\mathsf{init}}, C, \Sigma, I, E)$ gives rise to an uncountable transition system $\mathcal{T}(\mathcal{A}) = (S, s_{\mathsf{init}}, M, \to)$ with the state space $S = L \times \mathbb{R}^C_{\geq 0}$, the initial state $s_{\mathsf{init}} = (\ell_{\mathsf{init}}, \mathbf{0}^C)$, set of actions $M = \mathbb{R}_{\geq 0} \times (\Sigma \cup \{\bot\})$ and the transition relation $\to \subseteq S \times M \times S$ defined as follows: for any action $a \in \Sigma$ and delay $d \geq 0$, we have that $((\ell, \nu), (d, a), (\ell', \nu')) \in \to$ if and only if there is some edge $(\ell, g, a, D, \ell') \in E$ such that $\nu + d \models g$, $\nu' = \mathsf{reset}_D(\nu + d)$, $\nu + d \models I(\ell)$ and $\nu' \models I(\ell')$; for any delay $d \geq 0$, $((\ell, \nu), (d, \bot), (\ell, \nu + d)) \in \to$ if and only if $\nu + d \models I(\ell)$. Let us note that the satisfaction set of clock constraints is convex: it is described by a conjunction of inequalities. Whenever $\nu \models I(\ell)$, the above conditions $\nu + d \models I(\ell)$ (the invariant holds after the delay) are equivalent to requiring $\nu + d' \models I(\ell)$ for all $0 \leq d' \leq d$ (the invariant holds at each intermediate time step).

A *move* is any pair in $\mathbb{R}_{\geq 0} \times (\Sigma \cup \{\bot\})$ (i.e., an action in the transition system). For any move $m = (d, a)$ and states s, $s' \in S$, we write $s \xrightarrow{m} s'$ or $s \xrightarrow{d,a} s'$ as shorthand for $(s, m, s') \in \to$. Moves of the form $(d, \bot)$ are called *delay moves*. We say a move m is enabled in a state s if there is some s' such that $s \xrightarrow{m} s'$. There is at most one successor per move in a state, as we do not allow two guards on edges labeled by the same action to be simultaneously satisfied.

A *path* in a TA $\mathcal{A}$ is a finite or infinite sequence $s_0(d_0, a_0)s_1 \ldots \in S(MS)^* \cup (SM)^\omega$ such that for all j, s_j is a state of $\mathcal{T}(\mathcal{A})$ and for all $j > 0$, $s_{j-1} \xrightarrow{d_{j-1}, a_{j-1}} s_j$ is a transition in $\mathcal{T}(\mathcal{A})$. A path is *initial* if $s_0 = s_{\mathsf{init}}$. For clarity, we write $s_0 \xrightarrow{d_0, a_0} s_1 \xrightarrow{d_1, a_1} \cdots$ instead of $s_0(d_0, a_0)s_1(d_1, a_1) \ldots$.

An infinite path $\pi = (\ell_0, \nu_0) \xrightarrow{d_0, a_0} (\ell_1, \nu_1) \ldots$ is *time-divergent* if the sequence $(\nu_j(\gamma))_{j \in \mathbb{N}}$ is not bounded from above. A path which is not time-divergent is called *time-convergent*; time-convergent paths are traditionally ignored in analysis of timed automata [1, 2] as they model unrealistic behavior. This includes ignoring *Zeno paths*, which are time-convergent paths along which infinitely many actions appear. We write $\mathsf{Paths}(\mathcal{A})$ for the set of paths of $\mathcal{A}$.

Priorities. A *priority function* is a function $p\colon L \to \{0, \dots, d-1\}$ with $d \leq |L|$. We use priority functions to express parity objectives. A *k-dimensional priority function* is a function $p\colon L \to \{0, \dots, d-1\}^k$ which assigns vectors of priorities to locations.

Timed games. We consider two player games played on TAs. We refer to the players as player 1 ($\mathcal{P}_1$) for the system and player 2 ($\mathcal{P}_2$) for the environment. We use the notion of timed automaton games of [20].

A *timed* (automaton) *game* (TG) is a tuple $\mathcal{G} = (\mathcal{A}, \Sigma_1, \Sigma_2)$ where $\mathcal{A} = (L, \ell_{\mathsf{init}}, C, \Sigma, I, E)$ is a TA and (Σ_1, Σ_2) is a partition of Σ. We refer to actions in Σ_i as $\mathcal{P}_i$ actions for $i \in \{1, 2\}$.

Recall a move is a pair $(d, a) \in \mathbb{R}_{\geq 0} \times (\Sigma \cup \{\bot\})$. Let S denote the set of states of $\mathcal{T}(\mathcal{A})$. In each state $s = (\ell, \nu) \in S$, the moves available to $\mathcal{P}_1$ are the elements of the set $M_1(s)$ where $M_1(s) = \{(d, a) \in \mathbb{R}_{\geq 0} \times (\Sigma_1 \cup \{\bot\}) \mid \exists\, s',\, s \xrightarrow{d,a} s'\}$ which contains moves with $\mathcal{P}_1$ actions and delay moves that are enabled in s. The set $M_2(s)$ is defined analogously with $\mathcal{P}_2$ actions. We write M_1 and M_2 for the set of all moves of $\mathcal{P}_1$ and $\mathcal{P}_2$ respectively.

At each state s along a play, both players simultaneously select a move $(d^{(1)}, a^{(1)}) \in M_1(s)$ and $(d^{(2)}, a^{(2)}) \in M_2(s)$. Intuitively, the fastest player gets to act and in case of a tie, the move is chosen non-deterministically. This is formalized by the *joint destination function* $\delta : S \times M_1 \times M_2 \to 2^S$, defined by

$$\delta(s, (d^{(1)}, a^{(1)}), (d^{(2)}, a^{(2)})) = \begin{cases} \{s' \in S \mid s \xrightarrow{d^{(1)}, a^{(1)}} s'\} & \text{if } d^{(1)} < d^{(2)} \\ \{s' \in S \mid s \xrightarrow{d^{(2)}, a^{(2)}} s'\} & \text{if } d^{(1)} > d^{(2)} \\ \{s' \in S \mid s \xrightarrow{d^{(i)}, a^{(i)}} s', i = 1, 2\} & \text{if } d^{(1)} = d^{(2)}. \end{cases}$$

For $m^{(1)} = (d^{(1)}, a^{(1)}) \in M_1$ and $m^{(2)} = (d^{(2)}, a^{(2)}) \in M_2$, we write $\mathsf{delay}(m^{(1)}, m^{(2)}) = \min\{d^{(1)}, d^{(2)}\}$ to denote the delay occurring when $\mathcal{P}_1$ and $\mathcal{P}_2$ play $m^{(1)}$ and $m^{(2)}$ respectively.

A play is defined similarly to a path: it is a finite or infinite sequence of the form $s_0(m_0^{(1)}, m_0^{(2)})s_1(m_1^{(1)}, m_1^{(2)}) \ldots \in S((M_1 \times M_2)S)^* \cup (S(M_1 \times M_2))^\omega$ where for all indices j, $m_j^{(i)} \in M_i(s_j)$ for $i \in \{1, 2\}$, and for $j > 0$, $s_j \in \delta(s_{j-1}, m_{j-1}^{(1)}, m_{j-1}^{(2)})$. A play is *initial* if $s_0 = s_{\mathsf{init}}$. For a finite play $\pi = s_0 \ldots s_n$, we set $\mathsf{last}(\pi) = s_n$. For an infinite play $\pi = s_0 \ldots$, we write $\pi_{|n} = s_0(m_0^{(0)}, m_0^{(1)}) \ldots s_n$. A play follows a path in the TA, but there need not be a unique path compatible with a play: if along a play, at the nth step, the moves of both players share the same delay and target state, either move can label the nth transition in a matching path.

Similarly to paths, an infinite play $\pi = (\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)}) \cdots$ is *time-divergent* if and only if $(\nu_j(\gamma))_{j \in \mathbb{N}}$ is not bounded from above. Otherwise, we say a play is *time-convergent*. We define the following sets: $\mathsf{Plays}(\mathcal{G})$ for the set of plays of $\mathcal{G}$; $\mathsf{Plays}_{\mathit{fin}}(\mathcal{G})$ for the set of finite plays of $\mathcal{G}$; $\mathsf{Plays}_\infty(\mathcal{G})$ for the set of time-divergent plays of $\mathcal{G}$. We also write $\mathsf{Plays}(\mathcal{G}, s)$ to denote plays starting in a state s of $\mathcal{T}(\mathcal{A})$.

Note that our games are built on *deterministic* TAs. From a modeling standpoint, this is not restrictive, as we can simulate a non-deterministic TA through the actions of $\mathcal{P}_2$.

Strategies. A strategy for $\mathcal{P}_i$ is a function describing which move a player should use based on a play history. Formally, a strategy for $\mathcal{P}_i$ is a function $\sigma_i\colon \mathsf{Plays}_{\mathit{fin}}(\mathcal{G}) \to M_i$ such that for all $\pi \in \mathsf{Plays}_{\mathit{fin}}(\mathcal{G})$, $\sigma_i(\pi) \in M_i(\mathsf{last}(\pi))$. This last condition requires that each move given by a strategy be enabled in the last state of a play.

A play $s_0(m_0^{(1)}, m_0^{(2)})s_1 \ldots$ is said to be consistent with a $\mathcal{P}_i$-strategy σ_i if for all indices j, $m_j^{(i)} = \sigma_i(\pi_{|j})$. Given a $\mathcal{P}_i$-strategy σ_i, we define $\mathsf{Outcome}_i(\sigma_i)$ (resp. $\mathsf{Outcome}_i(\sigma_i, s)$) to be the set of plays (resp. set of plays starting in state s) consistent with σ_i.

Objectives. An objective represents the property we desire on paths of a TA or a goal of a player in a TG. Formally, we define an *objective* as a set $\Psi \subseteq \mathsf{Paths}(\mathcal{A})$ of infinite paths (when studying TAs) or a set $\Psi \subseteq \mathsf{Plays}(\mathcal{G})$ of infinite plays (when studying TGs). An objective is *state-based* (resp. *location-based*) if it depends solely on the sequence of states (resp. of locations) in a path or play. Any location-based objective is state-based.

▶ Remark 1. In the sequel, we present objectives exclusively as sets of plays. Definitions for paths are analogous as all the objectives defined hereafter are state-based.

We use the following classical location-based objectives. The *safety* objective for a set F of locations is the set of plays that never visit a location in F. The *co-Büchi* objective for a set F of locations consists of plays traversing locations in F finitely often. The *parity* objective for a priority function p over the set of locations requires that the smallest priority seen infinitely often is even.

Fix F a set of locations and p a priority function. The aforementioned objectives are formally defined as follows:

- $\mathsf{Safe}(F) = \{(\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)}) \ldots \in \mathsf{Plays}(\mathcal{G}) \mid \forall n,\ \ell_n \notin F\}$;
- $\mathsf{coBüchi}(F) = \{(\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)}) \ldots \in \mathsf{Plays}(\mathcal{G}) \mid \exists j,\ \forall n \geq j,\ \ell_n \notin F\}$;
- $\mathsf{Parity}(p) = \{(\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)}) \ldots \in \mathsf{Plays}(\mathcal{G}) \mid (\liminf_{n\to\infty} p(\ell_n)) \bmod 2 = 0\}$.

Winning conditions. In games, we distinguish objectives and *winning conditions.* We adopt the definition of [20]. Let Ψ be an objective. It is desirable to have victory be achieved in a physically meaningful way: for example, it is unrealistic to have a safety objective be achieved by stopping time. This motivates a restriction to time-divergent plays. However, this requires $\mathcal{P}_1$ to force the divergence of plays, which is not reasonable, as $\mathcal{P}_2$ can stall using delays with zero time units. Thus we also declare winning time-convergent plays where $\mathcal{P}_1$ is *blameless.* Let $\mathsf{Blameless}_1$ denote the set of $\mathcal{P}_1$-blameless plays, which we define in the following way.

Let $\pi = s_0(m_0^{(1)}, m_0^{(2)})s_1 \ldots$ be a (possibly finite) play. We say $\mathcal{P}_1$ is *not responsible* (or not to be blamed) for the transition at step n in π if either $d_n^{(2)} < d_n^{(1)}$ ($\mathcal{P}_2$ is faster) or $d_n^{(1)} = d_n^{(2)}$ and $s_n \xrightarrow{d_n^{(1)}, a_n^{(1)}} s_{n+1}$ does not hold in $\mathcal{T}(\mathcal{A})$ ($\mathcal{P}_2$'s move was selected and did not have the same target state as $\mathcal{P}_1$'s) where $m_n^{(i)} = (d_n^{(i)}, a_n^{(i)})$ for $i \in \{1, 2\}$. The set $\mathsf{Blameless}_1$ is formally defined as the set of infinite plays π such that there is some j such that for all $n \geq j$, $\mathcal{P}_1$ is not responsible for the transition at step n in π.

Given an objective Ψ, we set the winning condition $\mathsf{WC}_1(\Psi)$ for $\mathcal{P}_1$ to be the set of plays

$$\mathsf{WC}_1(\Psi) = (\Psi \cap \mathsf{Plays}_\infty(\mathcal{G})) \cup (\mathsf{Blameless}_1 \setminus \mathsf{Plays}_\infty(\mathcal{G})).$$

Winning conditions for $\mathcal{P}_2$ are defined by exchanging the roles of the players in the former definition. We consider that the two players are adversaries and have opposite objectives, Ψ and $\neg\Psi$ (shorthand for $\mathsf{Plays}(\mathcal{G}) \setminus \Psi$). There may be plays π such that $\pi \notin \mathsf{WC}_1(\Psi)$ and $\pi \notin \mathsf{WC}_2(\neg\Psi)$, e.g., any time-convergent play in which neither player is blameless.

A *winning strategy* for $\mathcal{P}_i$ for an objective Ψ from a state s_0 is a strategy σ_i such that $\mathsf{Outcome}_i(\sigma_i, s_0) \subseteq \mathsf{WC}_i(\Psi)$.

Decision problems. We consider two different problems for an objective Ψ. The first is the *verification problem* for Ψ, which asks given a timed automaton whether all *time-divergent initial* paths satisfy the objective. Second is the *realizability problem*, which asks whether in a timed automaton game with objective Ψ, $\mathcal{P}_1$ has a winning strategy from the initial state.

3 Window objectives

We consider the *fixed window parity* and *direct fixed window parity* problems from [14] and adapt the discrete-time requirements from their initial formulation to dense-time requirements for TAs and TGs. Intuitively, a direct fixed window parity objective for some bound λ requires that at all points along a play or a path, we see a window of size less than λ in which the smallest priority is even. The (non-direct) window parity objective requires that the direct objective holds for some suffix. In the sequel, we drop "fixed" from the name of these objectives.

In this section, we formalize the timed window parity objective in TGs as sets of plays. The definition for paths of TAs is analogous (see Rmk. 1). First, we define the *timed good window objective*, which formalizes the notion of good windows. Then we introduce the timed window parity objective and its direct variant. We compare these objectives to the parity objective and argue that satisfying a window objective implies satisfying a parity objective, and that window objectives do not coincide with parity objectives in general, via an example. We conclude this section by presenting some useful properties of this objective.

For this entire section, we fix a TG $\mathcal{G} = (\mathcal{A}, \Sigma_1, \Sigma_2)$ where $\mathcal{A} = (L, \ell_{\mathsf{init}}, C, \Sigma_1 \cup \Sigma_2, I, E)$, a priority function $p\colon L \to \{0, \dots, d-1\}$ and a bound $\lambda \in \mathbb{N} \setminus \{0\}$ on the size of windows.

3.1 Definitions

Good windows. A window objective is based on a notion of good windows. Intuitively, a good window for the parity objective is a fragment of a play in which less than λ time units pass and the smallest priority of the locations appearing in this fragment is even.

The timed good window objective encompasses plays in which there is a good window at the start of the play. We formally define the *timed good window (parity) objective* as the set

$$\begin{aligned}\mathsf{TGW}(p, \lambda) = \big\{(\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)}) \ldots \in \mathsf{Plays}(\mathcal{G}) \mid \exists j \in \mathbb{N},\ \min_{0 \le k \le j} p(\ell_k) \bmod 2 = 0 \\ \wedge\, \nu_j(\gamma) - \nu_0(\gamma) < \lambda\big\}.\end{aligned}$$

The timed good window objective is a state-based objective.

We introduce some terminology related to windows. Let $\pi = (\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)})(\ell_1, \nu_1) \ldots$ be an infinite play. We say that the window opened at step n *closes* at step j if $\min_{n \le k \le j} p(\ell_k)$ is even and for all $n \le j' < j$, $\min_{n \le k \le j'} p(\ell_k)$ is odd. Note that, in this case, we must have $\min_{n \le k \le j} p(\ell_k) = p(\ell_j)$. In other words, a window closes when an even priority smaller than all other priorities in the window is encountered. The window opened at step n is said to *close immediately* if $p(\ell_n)$ is even.

If a window does not close within λ time units, we refer to it as a *bad* window: the window opened at step n is a bad window if there is some $j^\star \ge n$ such that $\nu_{j^\star}(\gamma) - \nu_n(\gamma) \ge \lambda$ and for all $j \ge n$, if $\nu_j(\gamma) - \nu_n(\gamma) < \lambda$, then $\min_{n \le k \le j} p(\ell_k)$ is odd.

Direct timed window objective. The direct window parity objective in graph games requires that every suffix of the play belongs to the good window objective. To adapt this objective to a dense-time setting, we must express that *at all times, we have a good window*. We require that this property holds not only at states which appear explicitly along plays, but also in the continuum between them (during the delay within a location). To this end, let us introduce a notation for suffixes of play.

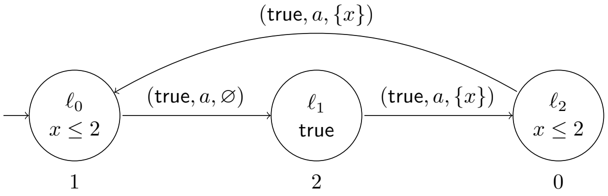


Figure 1 Timed automaton $\mathcal{A}$. Edges are labeled with triples guard-action-resets. Priorities are beneath locations. The initial state is denoted by an incoming arrow with no origin.

Let $\pi = (\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)})(\ell_1, \nu_1) \ldots \in \mathsf{Plays}(\mathcal{G})$ be a play. For all $i \in \{1, 2\}$ and all $n \in \mathbb{N}$, write $m_n^{(i)} = (d_n^{(i)}, a_n^{(i)})$ and $d_n = \mathsf{delay}(m_n^{(1)}, m_n^{(2)}) = \nu_{n+1}(\gamma) - \nu_n(\gamma)$. For any $n \in \mathbb{N}$ and $d \in [0, d_n]$, let $\pi_{n\rightarrow}^{+d}$ be the *delayed suffix* of π starting in position n delayed by d time units, defined as $\pi_{n\rightarrow}^{+d} = (\ell_n, \nu_n + d)((d_n^{(1)} - d, a_n^{(1)}), (d_n^{(2)} - d, a_n^{(2)}))(\ell_{n+1}, \nu_{n+1})(m_{n+1}^{(1)}, m_{n+1}^{(2)}) \ldots$ If $d = 0$, we write $\pi_{n\rightarrow}$ rather than $\pi_{n\rightarrow}^{+0}$.

Using the notations above, we define the *direct timed window (parity) objective* as the set

$$\mathsf{DTW}(p, \lambda) = \{\pi \in \mathsf{Plays}(\mathcal{G}) \mid \forall n \in \mathbb{N}, \forall d \in [0, d_n], \pi_{n\rightarrow}^{+d} \in \mathsf{TGW}(p, \lambda)\}.$$

The direct timed window objective is state-based: the timed good window objective is state-based and the delays d_n are encoded in states (by clock γ), thus all conditions in the definition of the direct timed window objective depend only the sequence of states of a play.

Timed window objective. We define the *timed window (parity) objective* as a prefix-independent variant of the direct timed window objective. Formally, we let

$$\mathsf{TW}(p, \lambda) = \{\pi \in \mathsf{Plays}(\mathcal{G}) \mid \exists n \in \mathbb{N}, \pi_{n\rightarrow} \in \mathsf{DTW}(p, \lambda)\}.$$

The timed window objective requires the direct timed window objective to hold from some point on. This implies that the timed window objective is state-based.

3.2 Comparison with parity objectives

Both the direct and non-direct timed window objectives reinforce the parity objective with time bounds. It can easily be shown that satisfying the direct timed window objective implies satisfying a parity objective. Any odd priority seen along a play in $\mathsf{DTW}(p, \lambda)$ is answered within λ time units by a smaller even priority. Therefore, should any odd priority appear infinitely often, it is followed by a smaller even priority. As the set of priorities is finite, there must be some smaller even priority appearing infinitely often. This in turn implies that the parity objective is fulfilled. It follows from prefix-independence of the parity objective, that satisfying the non-direct timed window objective implies satisfying the parity objective.

However, in some cases, the timed window objectives may not hold even though the parity objective holds. For simplicity, we provide an example on a TA, rather than a TG. Consider the timed automaton $\mathcal{A}$ depicted in Fig. 1.

All time-divergent paths of $\mathcal{A}$ satisfy the parity objective. We can classify time-divergent paths in two families: either ℓ_2 is visited infinitely often, or from some point on only delay transitions are taken in ℓ_1. In the former case, the smallest priority seen infinitely often is 0 and in the latter case, it is 2.

However, there is a path π such that for all window sizes $\lambda \in \mathbb{N} \setminus \{0\}$, π violates the direct and non-direct timed window objectives. Initialize n to 1. This path can be described by the following loop: play action a in ℓ_0 with delay 0, followed by action a with delay n in ℓ_1 and action a in ℓ_2 with delay 0, increase n by 1 and repeat. The window opened in ℓ_0 only closes when location ℓ_2 is entered. At the n-th step of the loop, this window closes after n time units. As we let n increase to infinity, there is no window size λ such that this path satisfies the direct and non-direct timed window objectives for λ.

This example demonstrates the interest of reinforcing parity objectives with time bounds; we can enforce that there is a bounded delay between an odd priority and a smaller even priority in a path.

3.3 Properties of window objectives

We present several properties of the timed window objective. First, we show that we need only check good windows for non-delayed suffixes $\pi_{n\to}$. Once this property is introduced, we move on to the inductive property of windows, which is the crux of the reduction in the next section. This inductive property states that when we close a window in less than λ time units all other windows opened in the meantime also close in less than λ time units.

The definition of the direct timed window objective requires checking uncountably many windows. This can be reduced to a countable number of windows: those opened when entering states appearing along a play. Let us explain why no information is lost through such a restriction. We rely on a timeline-like visual representation given in Fig. 2. Consider a window that does not close immediately and is opened in some state of the play delayed by d time units, of the form $(\ell_n, \nu_n + d)$ (depicted by the circle at the start of the bottom line of Fig. 2). This implies that the priority of ℓ_n is odd, otherwise this window would close immediately. Assume the window opened at step n closes at step j (illustrated by the middle line of the figure) in less than λ time units. As the priority of ℓ_n is odd, we must have $j \geq n+1$ (i.e., the window opened at step n is still open as long as ℓ_n is not left). These lines cover the same locations, i.e., the set of locations appearing along the time-frame given by both the dotted and dashed lines coincide. Thus, the window opened d time units after step n closes in at most $\lambda - d$ time units, at the same time as the window opened at step n.

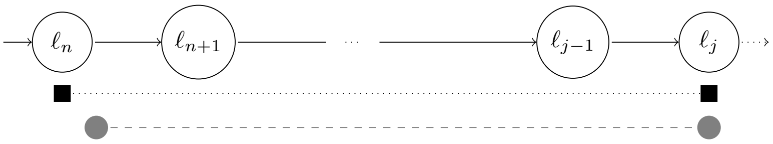


Figure 2 A timeline representation of a play. Circles with labels indicate entry in a location. The dotted line underneath represents a window opened at step n and closed at step j and the dashed line underneath the window opened d time units after step n.

▶ **Lemma 2.** *Let* $\pi = (\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)}) \ldots \in \mathsf{Plays}(\mathcal{G})$ *and* $n \in \mathbb{N}$. *Let* d_n *denote* $\mathsf{delay}(m_n^{(1)}, m_n^{(2)})$. *Then* $\pi_{n\to} \in \mathsf{TGW}(p, \lambda)$ *if and only if for all* $d \in [0, d_n]$, $\pi_{n\to}^{+d} \in \mathsf{TGW}(p, \lambda)$. *Furthermore,* $\pi \in \mathsf{DTW}(p, \lambda)$ *if and only if for all* $n \in \mathbb{N}$, $\pi_{n\to} \in \mathsf{TGW}(p, \lambda)$.

In turn-based games on graphs, window objectives exhibit an inductive property: when a window closes, all subsequently opened windows close (or were closed earlier) [14]. This is also the case for the timed variant. A window closes when an even priority smaller than all priorities seen in the window is encountered. This priority is also smaller than priorities in all windows opened in the meantime, therefore they must close at this point (if they are not yet closed). We state this property only for windows opened at steps along the run and neglect the continuum in between due to Lem. 2.

▶ **Lemma 3** (Inductive property). *Let* $\pi = (\ell_0, \nu_0)(m_0^{(1)}, m_0^{(2)})(\ell_1, \nu_1) \ldots \in \mathsf{Plays}(\mathcal{G})$*. Let* $n \in \mathbb{N}$*. Assume the window opened at step* n *closes at step* j*. Then, for all* $n \leq i \leq j$*,* $\pi_{i\to} \in \mathsf{TGW}(p, \lambda)$*.*

It follows from this inductive property that it suffices to keep track of one window at a time when checking whether a play satisfies the (direct) timed window objective.

4 Reduction

We establish that the realizability (resp. verification) problem for the direct/non-direct timed window parity objective can be reduced to the realizability (resp. verification) problem for safety/co-Büchi objectives on an expanded TG (resp. TA). Our reduction uses the same construction of an expanded TA for both the verification and realizability problems. A state of the expanded TA describes the status of a window, allowing the detection of bad windows. Due to space constraints, we only describe the reduction here and sketch the main technical hurdles along the way. The full approach, including intermediate results, is presented in the full version of our paper [23].

The sketch is as follows. First, we describe how a TA can be expanded with window-related information. Second, we show that time-divergent plays in a TG and its expansion can be related, by constructing two (non-bijective) mappings, in a manner such that a time-divergent play in the base TG satisfies the direct/non-direct timed window parity objective if and only if its related play in the expanded TG satisfies the safety/co-Büchi objective. These results developed for plays are (indirectly) applied to paths in order to show the correctness of the reduction for the verification problem. Third, we establish that the mappings developed in the second part can be leveraged to translate strategies in TGs, and prove that the presented translations preserve winning strategies, proving correctness of the reduction for TGs.

For this section, we fix a TG $\mathcal{G} = (\mathcal{A}, \Sigma_1, \Sigma_2)$ with TA $\mathcal{A} = (L, \ell_{\mathsf{init}}, C, \Sigma, I, E)$, a priority function p and a bound λ on the size of windows.

Encoding the objective in an automaton. The inductive property (Lem. 3) implies that it suffices to keep track of one window at a time when checking a window objective. Following this, we encode the status of a window in the TA.

A window can be summarized by two characteristics: the lowest priority within it and for how long it has been open. To keep track of the first trait, we encode the lowest priority seen in the current window in locations of the TA. An expanded location is a pair (ℓ, q) where $q \in \{0, \ldots, d-1\}$; the number q represents the smallest priority in the window currently under consideration. We say a pair (ℓ, q) is an even (resp. odd) location if q is even (resp. odd). To measure how long a window is opened, we use an additional clock $z \notin C$ that does not appear in $\mathcal{A}$. This clock is reset whenever a new window opens or a bad window is detected.

The focus of the reduction is over time-divergent plays. Some time-convergent plays may violate a timed good window objective without ever seeing a bad window, e.g., when time does not progress up to the supplied window size. Along time-divergent plays however, the lack of a good window at any point equates to the presence of a bad window. We encode the (resp. direct) timed window objective as a co-Büchi (resp. safety) objective. Locations to avoid in both cases indicate bad windows and are additional expanded locations (ℓ, bad), referred to as *bad locations*. We introduce two new actions β_1 and β_2, one per player, for entering and exiting bad locations. While only the action β_1 is sufficient for the reduction to be correct, introducing two actions allows for a simpler correctness proof in the case of TGs; we can exploit the fact that $\mathcal{P}_2$ can enter and exit bad locations.

It remains to discuss how the initial location, edges and invariants of an expanded TA are defined. We discuss edges and invariants for each type of expanded location, starting with even locations, then odd locations and finally bad locations. Each rule we introduce hereafter is followed by an application on an example. We depict the TA of Fig. 1 and the reachable fragment of its expansion in Fig. 3 and use these TAs for our example. For this explanation, we use the terminology of TAs (paths) rather than that of TGs (plays).

The initial location of an expanded TA encodes the window opened at the start of an initial path of the original TA. This window contains only a single priority, that is the priority of the initial location of the original TA. Thus, the initial location of the expanded TA is the expanded location $(\ell_{\mathsf{init}}, p(\ell_{\mathsf{init}}))$. In our example, the initial location of the expanded TA is $(\ell_0, 1)$.

Even expanded locations encode windows that are closed and do not need to be monitored anymore. Therefore, the invariant of an even expanded location is unchanged from the invariant of the original location in the original TA. Similarly, we do not add any additional constraints on the edges leaving even expanded locations. Leaving an even expanded location means opening a new window: any edge leaving an even expanded location has an expanded location of the form $(\ell, p(\ell))$ as its target ($p(\ell)$ is the only priority in the new window) and resets z to start measuring the size of the new window. For example, the edge from $(\ell_2, 0)$ to $(\ell_0, 1)$ of the expanded TA of Fig. 3 is derived from the edge from ℓ_2 to ℓ_0 of the original TA.

Odd expanded locations represent windows that are still open. The clock z measures how long a window has been opened. If z reaches λ in an odd expanded location, that equates to a bad window in the original TA. In this case, we force time-divergent paths of the expanded TA to visit a bad location. This is done in three steps. We strengthen the invariant of odd expanded locations to prevent z from exceeding λ. We also disable the edges that leave odd expanded locations and do not go to a bad location whenever $z = \lambda$ holds, by reinforcing the guards of such edges by $z < \lambda$. Finally, we include two edges to a bad location (one per additional action β_1 and β_2), which can only be used whenever there is a bad window, i.e., when $z = \lambda$. In the case of our example, if z reaches λ in $(\ell_0, 1)$, we redirect the path to location (ℓ_0, bad), indicating a window has not closed in time in ℓ_0. When z reaches λ in $(\ell_0, 1)$, no more non-zero delays are possible, the edge from $(\ell_0, 1)$ to $(\ell_1, 1)$ is disabled and only the edges to (ℓ_0, bad) are enabled.

When leaving an odd expanded location using an edge, assuming we do not go to a bad location, the smallest priority of the window has to be updated. The new smallest priority is the minimum between the smallest priority of the window prior to traversing the edge and the priority of the target location. In our example for instance, the edge from $(\ell_1, 1)$ to $(\ell_2, 0)$ is derived from the edge from ℓ_1 to ℓ_2 in the original TA. As the priority of ℓ_2 is 0 and is smaller than the current smallest priority of the window encoded by location $(\ell_1, 1)$, the smallest priority of the window is updated to $0 = \min\{1, p(\ell_2)\}$ when traversing the edge. Note that we do not reset z despite the encoded window closing upon entering $(\ell_2, 0)$: the value of z does not matter while in even locations, thus there is no need for a reset when closing the window.

A bad location (ℓ, bad) is entered whenever a bad window is detected while in location ℓ. Bad locations are equipped with the invariant $z = 0$ preventing the passage of time. In this way, for time-divergent paths, a new window is opened immediately after a bad window is detected. For each additional action β_1 and β_2, we add an edge exiting the bad location. Edges leaving a bad location (ℓ, bad) have as their target the expanded location $(\ell, p(\ell))$; we reopen a window in the location in which a bad window was detected. The clock z is not reset by these edges, as it was reset prior to entering the bad location and the invariant

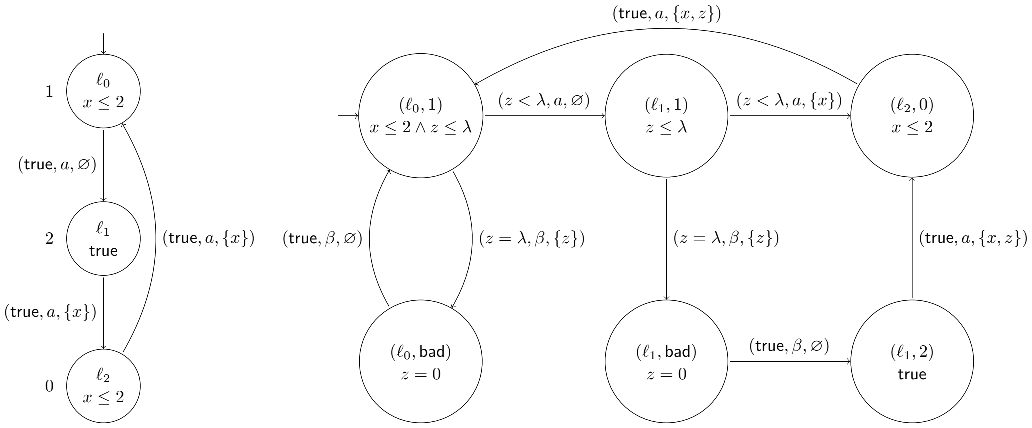


Figure 3 The TA of Fig. 1 (left) and the reachable fragment of its expansion (right). We write β for actions β_1 and β_2.

$z = 0$ prevents any non-zero delay in the bad location. For instance, the edges from (ℓ_1, bad) to $(\ell_1, 2)$ in our example represent that when reopening while in location ℓ_1, the smallest priority of this window is $p(\ell_1) = 2$.

The expansion depends on the priority function p and the bound on the size of windows λ. Therefore, we write $\mathcal{A}(p, \lambda)$ for the expansion. The formal definition of $\mathcal{A}(p, \lambda)$ follows.

▶ **Definition 4.** *Given a TA $\mathcal{A} = (L, \ell_{\mathsf{init}}, C, \Sigma, I, E)$, the TA $\mathcal{A}(p, \lambda)$ is defined to be the TA $(L', \ell'_{\mathsf{init}}, C', \Sigma', I', E')$ such that*

- *$L' = L \times (\{0, \dots, d-1\} \cup \{\mathsf{bad}\})$;*
- *$\ell'_{\mathsf{init}} = (\ell_{\mathsf{init}}, p(\ell_{\mathsf{init}}))$;*
- *$C' = C \cup \{z\}$ where $z \notin C$ is a new clock;*
- *$\Sigma' = \Sigma \cup \{\beta_1, \beta_2\}$ is an expanded set of actions with special actions β_1, $\beta_2 \notin \Sigma$ for bad locations;*
- *$I'(\ell, q) = I(\ell)$ for all $\ell \in L$ and even $q \in \{0, \dots, d-1\}$, $I'(\ell, q) = (I(\ell) \wedge z \leq \lambda)$ for all $\ell \in L$ and odd $q \in \{0, \dots, d-1\}$, and $I'(\ell, \mathsf{bad}) = (z = 0)$ for all $\ell \in L$;*
- *the set of edges E' of $\mathcal{A}(p, \lambda)$ is the smallest set satisfying the following rules:*
  - *if q is even and $(\ell, g, a, D, \ell') \in E$, then $((\ell, q), g, a, D \cup \{z\}, (\ell', p(\ell'))) \in E'$;*
  - *if q is odd and $(\ell, g, a, D, \ell') \in E$, then $((\ell, q), (g \wedge z < \lambda), a, D, (\ell', \min\{q, p(\ell')\})) \in E'$;*
  - *for all $\ell \in L$, odd q and $\beta \in \{\beta_1, \beta_2\}$, $((\ell, q), (z = \lambda), \beta, \{z\}, (\ell, \mathsf{bad})) \in E'$ and $((\ell, \mathsf{bad}), \mathsf{true}, \beta, \varnothing, (\ell, p(\ell)) \in E'$.*

For a TG $\mathcal{G} = (\mathcal{A}, \Sigma_1, \Sigma_2)$, we set $\mathcal{G}(p, \lambda) = (\mathcal{A}(p, \lambda), \Sigma_1 \cup \{\beta_1\}, \Sigma_2 \cup \{\beta_2\})$.

We write $(\ell, q, \bar{\nu})$ for states of $\mathcal{T}(\mathcal{A}(p, \lambda))$ instead of $((\ell, q), \bar{\nu})$, for conciseness. The bar over the valuation is a visual indicator of the different domain. We write $\mathsf{Bad} = L \times \{\mathsf{bad}\}$ for the set of bad locations.

Expanding and projecting plays. A bijection between the set of plays of $\mathcal{G}$ and the set of plays of $\mathcal{G}(p, \lambda)$ cannot be achieved naturally due to the additional information encoded in the expanded TA, notably the presence of bad locations. We illustrate this by showing there are some plays of $\mathcal{G}(p, \lambda)$ that are intuitively indistinguishable if seen as plays of $\mathcal{G}$.

Consider the initial location ℓ_{init} of $\mathcal{G}$, and assume that its priority is odd and its invariant is true. Consider the initial play $\bar{\pi}_1$ of $\mathcal{G}(p, \lambda)$ where the actions β_i are used by both players with a delay of λ at the start of the play and then only delay moves are taken in the reached bad location, i.e., $\bar{\pi}_1 = (\ell, p(\ell), \mathbf{0}^{C\cup\{z\}})((\lambda, \beta_1), (\lambda, \beta_2))\big((\ell, \mathsf{bad}, \bar{\nu})((0, \bot), (0, \bot))\big)^\omega$, where $\bar{\nu}(x) = \lambda$ for all $x \in C$ and $\bar{\nu}(z) = 0$. As the actions β_i and z do not exist in $\mathcal{G}$, $\bar{\pi}_1$ cannot be discerned from the similar play $\bar{\pi}_2$ of $\mathcal{G}(p, \lambda)$ where instead of using the actions β_i, delay moves were used instead, i.e., $\bar{\pi}_2 = (\ell, p(\ell), \mathbf{0}^{C\cup\{z\}})((\lambda, \bot), (\lambda, \bot))((\ell, p(\ell), \bar{\nu}')((0, \bot), (0, \bot)))^\omega$ with $\bar{\nu}'(x) = \lambda$ for all $x \in C \cup \{z\}$.

This motivates using two mappings instead of a bijection. Using the expansion mapping $\mathsf{Ex}\colon \mathsf{Plays}(\mathcal{G}) \to \mathsf{Plays}(\mathcal{G}(p, \lambda))$ and projection mapping $\mathsf{Pr}\colon \mathsf{Plays}(\mathcal{G}(p, \lambda)) \to \mathsf{Plays}(\mathcal{G})$ defined in the full paper [23], we obtain the following equivalence.

▶ **Theorem 5.** *Let $\mathcal{A} = (L, \ell_{\mathsf{init}}, C, \Sigma, I, E)$ be a TA, p a priority function and $\lambda \in \mathbb{N} \setminus \{0\}$. All time-divergent paths of $\mathcal{A}$ satisfy the (resp. direct) timed window objective if and only if all time-divergent paths of $\mathcal{A}(p, \lambda)$ satisfy the co-Büchi (resp. safety) objective over bad locations.*

Translating strategies. We translate $\mathcal{P}_1$-strategies of the expanded TG to $\mathcal{P}_1$-strategies of the original TG by evaluating the strategy on the expanded TG on expansions of plays provided by the expansion mapping and by replacing any occurrences of β_1 by $\bot$. The fact that translating a winning strategy of the expanded TG this way yields a winning strategy of the original TG is not straightforward. When we translate a winning strategy $\bar{\sigma}$ of $\mathcal{G}(p, \lambda)$ to a strategy σ of $\mathcal{G}$, the expansion of an outcome π of σ may not be consistent with $\bar{\sigma}$, e.g., moves of the form (d, β_1) may appear along $\mathsf{Ex}(\pi)$ in places where $\bar{\sigma}$ suggests otherwise, therefore, it cannot be inferred from the fact that $\bar{\sigma}$ is winning that $\mathsf{Ex}(\pi)$ satisfies the winning condition. We circumvent this issue by constructing another play $\bar{\pi}$ in parallel that is consistent with $\bar{\sigma}$ and shares the same sequence of states as $\mathsf{Ex}(\pi)$. This ensures, by the properties of the expansion mapping, that all time-divergent outcomes of σ are also winning. Furthermore, if the constructed play $\bar{\pi}$ is $\mathcal{P}_1$-blameless and time-convergent, then π also is $\mathcal{P}_1$-blameless, ensuring that all time-convergent outcomes of σ are winning.

In the other direction, we translate $\mathcal{P}_1$-strategies of the original TG to strategies of the expanded TG using the projection mapping and by replacing moves that are illegal in the expansion with suitable moves of the form (d, β_1). The technical difficulties for this translation are similar to those encountered in the first one and are handled similarly. From these two translations, we obtain the following equivalence.

▶ **Theorem 6.** *Let s_{init} be the initial state of $\mathcal{G}$ and $\bar{s}_{\mathsf{init}}$ be the initial state of $\mathcal{G}(p, \lambda)$. There is a winning strategy σ for $\mathcal{P}_1$ for the objective* $\mathsf{TW}(p, \lambda)$ *(resp.* $\mathsf{DTW}(p, \lambda)$*) from s_{init} in $\mathcal{G}$ if and only if there is a winning strategy $\bar{\sigma}$ for $\mathcal{P}_1$ for the objective* $\mathsf{coB\ddot{u}chi}(\mathsf{Bad})$ *(resp.* $\mathsf{Safe}(\mathsf{Bad})$*) from $\bar{s}_{\mathsf{init}}$ in $\mathcal{G}(p, \lambda)$.*

Multi-dimensional objectives. The construction of the expanded TA $\mathcal{A}(p, \lambda)$ can be extended to handle *generalized (resp. direct) timed window objectives*, defined as conjunctions of several (resp. direct) timed window objectives. Generalized objectives are given by k-dimensional priority functions and vectors $\lambda \in (\mathbb{N} \setminus \{0\})^k$ of bounds on the size of windows for each dimension. To keep track of the windows for each dimension, locations are expanded using k-dimensional vectors and one new clock per objective is added to the expanded automaton. The expanded TA in this case is of size exponential in k. The objective on the expansion is a co-Büchi or safety objective, even for multiple objectives. Due to space constraints, we omit the full generalization, which can be found in the full paper [23].

5 Algorithms and complexity

This section outlines algorithms for solving the verification and realizability problems for generalized (resp. direct) timed window parity objectives. We consider the general multi-dimensional setting and we denote by k the number of timed window parity objectives under consideration. Technical details and a comparison with timed parity games can be found in the full paper [23].

Algorithms. Our solution to the verification and realizability problem for generalized (resp. direct) timed window objectives consists of a reduction to a co-Büchi (resp. safety) objective on an expanded automaton. The following lemma establishes the time necessary for the reduction.

▶ **Lemma 7.** *Let $\mathcal{A} = (L, \ell_{\mathsf{init}}, C, \Sigma, I, E)$ be a TA. Let $p\colon L \to \{0, \ldots, d-1\}^k$ be a k-dimensional priority function and $\lambda \in (\mathbb{N} \setminus \{0\})^k$ be a vector of bounds on window sizes. The expanded TA $\mathcal{A}(p, \lambda) = (L', \ell'_{\mathsf{init}}, C', \Sigma', I', E')$ can be computed in time exponential in k and polynomial in the size of L, the size of E, the size of C, d, the length of the encoding of the clock constraints of $\mathcal{A}$ and the encoding of λ.*

Let $\mathcal{G}$ be a TG. To solve the realizability problem on the expanded TG $\mathcal{G}(p, \lambda)$, we use the algorithm of [20], which checks the existence of a winning strategy for $\mathcal{P}_1$ for location-based objectives specified by a deterministic parity automaton. Using this algorithm, we can check the existence of a winning strategy for $\mathcal{P}_1$ for the co-Büchi (resp. safety) objective on $\mathcal{G}(p, \lambda)$ in time $\mathcal{O}\big(\big(|L|^2 \cdot (d^k + 1)^2 \cdot (|C| + k)! \cdot 2^{|C|+k} \prod_{x \in C}(2c_x + 1) \cdot \prod_{1 \leq i \leq k}(2\lambda_i + 1)\big)^4\big)$, where L is the set of locations of $\mathcal{G}$, C the set of clocks of $\mathcal{G}$ and c_x is the largest constant to which $x \in C$ is compared to in $\mathcal{G}$ (recall that λ_i refers to the bound on window sizes for the ith dimension). This establishes membership of the realizability problem in EXPTIME as the construction of the expanded game can be done in exponential time by Lem. 7.

Let us move on to the verification problem. For one dimension, the verification problem for the (resp. direct) timed window objective is reducible in polynomial time to the verification problem for co-Büchi (resp. safety) objectives by Lem. 7. This establishes membership in PSPACE for the case of a single dimension. For multiple dimensions, the single-dimensional case can be used as an oracle to check that, for each individual objective, all time-divergent paths belong to the objective. This shows membership of the verification problem for generalized objectives in $\mathsf{P}^{\mathsf{PSPACE}} = \mathsf{PSPACE}$ [6].

Lower bounds. Lower bounds are established by reducing the verification/realizability problem for safety objectives to the verification/realizability problem for (resp. direct) timed window parity objectives. The verification problem for safety objectives is PSPACE-complete [2] and the realizability problem for safety objectives is EXPTIME-complete [21].

The reduction is as follows. Given a TA $\mathcal{A}$ and a set F of locations to avoid, we construct a TA $\mathcal{A}'$ obtained by extending locations of $\mathcal{A}$ with a Boolean indicating whether F has been visited. We assign an odd priority to expanded locations that indicate F has been visited, such that windows can no longer close if F has been visited.

Complexity wrap-up. We summarize our complexity results in the following theorem.

▶ **Theorem 8.** *The verification problem for the (direct) generalized timed window parity objective is PSPACE-complete and the realizability problem for the (direct) generalized timed window parity objective is EXPTIME-complete.*

References

1 Rajeev Alur, Costas Courcoubetis, and David L. Dill. Model-checking in dense real-time. *Inf. Comput.*, 104(1):2–34, 1993. doi:10.1006/inco.1993.1024.

2 Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994. doi:10.1016/0304-3975(94)90010-8.

3 Christel Baier. Reasoning about cost-utility constraints in probabilistic models. In Mikolaj Bojanczyk, Slawomir Lasota, and Igor Potapov, editors, *Reachability Problems – 9th International Workshop, RP 2015, Warsaw, Poland, September 21-23, 2015, Proceedings*, volume 9328 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 2015. doi:10.1007/978-3-319-24537-9_1.

4 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

5 Christel Baier, Joachim Klein, Sascha Klüppelholz, and Sascha Wunderlich. Weight monitoring with linear temporal logic: complexity and decidability. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14–18, 2014*, pages 11:1–11:10. ACM, 2014. doi:10.1145/2603088.2603162.

6 Theodore P. Baker, John Gill, and Robert Solovay. Relativizations of the P =? NP question. *SIAM J. Comput.*, 4(4):431–442, 1975. doi:10.1137/0204037.

7 Raphaël Berthon, Mickael Randour, and Jean-François Raskin. Threshold constraints with guarantees for parity objectives in Markov decision processes. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 121:1–121:15. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.121.

8 Benjamin Bordais, Shibashis Guha, and Jean-François Raskin. Expected window mean-payoff. In Arkadev Chattopadhyay and Paul Gastin, editors, *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2019, December 11-13, 2019, Bombay, India*, volume 150 of *LIPIcs*, pages 32:1–32:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.FSTTCS.2019.32.

9 Patricia Bouyer, Thomas Brihaye, Mickael Randour, Cédric Rivière, and Pierre Vandenhove. Decisiveness of stochastic systems and its application to hybrid models. In Jean-François Raskin and Davide Bresolin, editors, *Proceedings 11th International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2020, Brussels, Belgium, September 21-22, 2020*, volume 326 of *EPTCS*, pages 149–165, 2020. doi:10.4204/EPTCS.326.10.

10 Tomás Brázdil, Krishnendu Chatterjee, Vojtech Forejt, and Antonín Kucera. Trading performance for stability in Markov decision processes. *J. Comput. Syst. Sci.*, 84:144–170, 2017. doi:10.1016/j.jcss.2016.09.009.

11 Tomás Brázdil, Vojtech Forejt, Antonín Kucera, and Petr Novotný. Stability in graphs and games. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, pages 10:1–10:14. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.10.

12 Thomas Brihaye, Florent Delgrange, Youssouf Oualhadj, and Mickael Randour. Life is random, time is not: Markov decision processes with window objectives. *Log. Methods Comput. Sci.*, 16(4), 2020. URL: https://lmcs.episciences.org/6975.

13 Véronique Bruyère, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. *Inf. Comput.*, 254:259–295, 2017. doi:10.1016/j.ic.2016.10.011.

14 Véronique Bruyère, Quentin Hautem, and Mickael Randour. Window parity games: an alternative approach toward parity games with time bounds. In Domenico Cantone and Giorgio Delzanno, editors, *Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016, Catania, Italy, 14-16 September 2016*, volume 226 of *EPTCS*, pages 135–148, 2016. doi:10.4204/EPTCS.226.10.

15 Véronique Bruyère, Quentin Hautem, and Jean-François Raskin. On the complexity of heterogeneous multidimensional games. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, pages 11:1–11:15. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.11.

16 Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, June 19-23, 2017*, pages 252–263. ACM, 2017. doi:10.1145/3055399.3055409.

17 Krishnendu Chatterjee, Laurent Doyen, Mickael Randour, and Jean-François Raskin. Looking at mean-payoff and total-payoff through windows. *Inf. Comput.*, 242:25–52, 2015. doi:10.1016/j.ic.2015.03.010.

18 Krishnendu Chatterjee, Thomas A. Henzinger, and Florian Horn. Finitary winning in omega-regular games. *ACM Trans. Comput. Log.*, 11(1):1:1–1:27, 2009. doi:10.1145/1614431.1614432.

19 Krishnendu Chatterjee, Thomas A. Henzinger, and Vinayak S. Prabhu. Timed parity games: Complexity and robustness. *Log. Methods Comput. Sci.*, 7(4), 2011. doi:10.2168/LMCS-7(4:8)2011.

20 Luca de Alfaro, Marco Faella, Thomas A. Henzinger, Rupak Majumdar, and Mariëlle Stoelinga. The element of surprise in timed games. In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR 2003 – Concurrency Theory, 14th International Conference, Marseille, France, September 3-5, 2003, Proceedings*, volume 2761 of *Lecture Notes in Computer Science*, pages 142–156. Springer, 2003. doi:10.1007/978-3-540-45187-7_9.

21 Thomas A. Henzinger and Peter W. Kopke. Discrete-time control for rectangular hybrid automata. *Theor. Comput. Sci.*, 221(1-2):369–392, 1999. doi:10.1016/S0304-3975(99)00038-9.

22 Paul Hunter, Guillermo A. Pérez, and Jean-François Raskin. Looking at mean payoff through foggy windows. *Acta Inf.*, 55(8):627–647, 2018. doi:10.1007/s00236-017-0304-7.

23 James C. A. Main, Mickael Randour, and Jeremy Sproston. Time flies when looking out of the window: Timed games with window parity objectives. *CoRR*, abs/2105.06686, 2021. arXiv:2105.06686.

24 Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems (an extended abstract). In Ernst W. Mayr and Claude Puech, editors, *STACS 95, 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, Germany, March 2-4, 1995, Proceedings*, volume 900 of *Lecture Notes in Computer Science*, pages 229–242. Springer, 1995. doi:10.1007/3-540-59042-0_76.

25 Mickael Randour. Automated synthesis of reliable and efficient systems through game theory: A case study. In *Proc. of ECCS 2012*, Springer Proceedings in Complexity XVII, pages 731–738. Springer, 2013. doi:10.1007/978-3-319-00395-5_90.

26 Stéphane Le Roux, Arno Pauly, and Mickael Randour. Extending finite-memory determinacy by Boolean combination of winning conditions. In Sumit Ganguly and Paritosh K. Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPIcs*, pages 38:1–38:20. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.38.

Arena-Independent Finite-Memory Determinacy in Stochastic Games

Patricia Bouyer
Université Paris-Saclay, CNRS, ENS Paris-Saclay,
Laboratoire Méthodes Formelles, 91190, Gif-sur-Yvette, France

Youssouf Oualhadj
Univ Paris Est Creteil, LACL, F-94010 Creteil, France

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

Pierre Vandenhove
F.R.S.-FNRS & UMONS – Université de Mons, Belgium
Université Paris-Saclay, CNRS, ENS Paris-Saclay,
Laboratoire Méthodes Formelles, 91190, Gif-sur-Yvette, France

Abstract

We study stochastic zero-sum games on graphs, which are prevalent tools to model decision-making in presence of an antagonistic opponent in a random environment. In this setting, an important question is the one of *strategy complexity*: what kinds of strategies are sufficient or required to play optimally (e.g., randomization or memory requirements)? Our contributions further the understanding of *arena-independent finite-memory (AIFM) determinacy*, i.e., the study of objectives for which memory is needed, but in a way that only depends on limited parameters of the game graphs. First, we show that objectives for which pure AIFM strategies suffice to play optimally also admit pure AIFM *subgame perfect* strategies. Second, we show that we can reduce the study of objectives for which pure AIFM strategies suffice in two-player stochastic games to the easier study of one-player stochastic games (i.e., Markov decision processes). Third, we *characterize* the sufficiency of AIFM strategies through two intuitive properties of objectives. This work extends a line of research started on deterministic games in [7] to stochastic ones.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and phrases two-player games on graphs, stochastic games, Markov decision processes, finite-memory determinacy, optimal strategies

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.26

Related Version *Full Version*: `https://arxiv.org/abs/2102.10104` [9]

Funding Research supported by F.R.S.-FNRS under Grant n° F.4520.18 (ManySynth), and ENS Paris-Saclay visiting professorship (M. Randour, 2019). This work has been partially supported by the ANR Project MAVeriQ (ANR-ANR-20-CE25-0012). Mickael Randour is an F.R.S.-FNRS Research Associate. Pierre Vandenhove is an F.R.S.-FNRS Research Fellow.

1 Introduction

Controller synthesis consists, given a system, an environment, and a specification, in automatically generating a controller of the system that guarantees the specification in the environment. This task is often studied through a game-theoretic lens: the system is a game, the controller is a player, the uncontrollable environment is its adversary, and the specification is a game objective [45]. A *game on graph* consists of a directed graph, called an *arena*, partitioned into two kinds of vertices: some of them are controlled by the system (called *player* 1) and the others by the environment (called *player* 2). Player 1 is given a

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 26; pp. 26:1–26:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

game objective (corresponding to the specification) and must devise a *strategy* (corresponding to the controller) to accomplish the objective or optimize an outcome. The strategy can be seen as a function that dictates the decisions to make in order to react to every possible chain of events. In case of uncertainty in the system or the environment, probability distributions are often used to model transitions in the game graph, giving rise to the *stochastic game* model. We study here *stochastic turn-based zero-sum games on graphs* [29], also called perfect-information stochastic games. We also discuss the case of *deterministic games*, which can be seen as a subcase of stochastic games in which only Dirac distributions are used in transitions.

Strategy complexity. A common question underlying all game objectives is the one of *strategy complexity*: how *complex* must optimal strategies be, and how *simple* can optimal strategies be? For each distinct game objective, multiple directions can be investigated, such as the need for randomization [19] (must optimal strategies make stochastic choices?), the need for memory [33, 34, 7] (how much information about the past must optimal strategies remember?), or what trade-offs exist between randomization and memory [16, 26, 41]. With respect to memory requirements, three cases are typically distinguished: *memoryless-determined objectives*, for which *memoryless* strategies suffice to play optimally; *finite-memory-determined objectives*, for which finite-memory strategies suffice (memory is then usually encoded as a deterministic finite automaton); and objectives for which infinite memory is required. High memory requirements (such as exponential memory and obviously infinite memory) are a major drawback when it comes to implementing controllers; hence specific approaches are often developed to look for *simple* strategies (e.g., [30]).

Many classical game objectives (reachability [29], Büchi and parity [22], discounted sum [48], energy [10]. . .) are memoryless-determined, both in deterministic and stochastic arenas. Nowadays, multiple general results allow for a more manageable proof for most of these objectives: we mention [35, 4, 1] for sufficient conditions in deterministic games, and [31, 32] for similar conditions in one-player and two-player stochastic games. One milestone for memoryless determinacy in deterministic games was achieved by Gimbert and Zielonka [33], who provide *two characterizations* of it: the first one states two necessary and sufficient conditions (called *monotony* and *selectivity*) for memoryless determinacy, and the second one states that memoryless determinacy in both players' *one-player games* suffices for memoryless determinacy in two-player games (we call this result the *one-to-two-player lift*). Together, these characterizations provide a theoretical and practical advance. On the one hand, monotony and selectivity improve the high-level understanding of what conditions well-behaved objectives verify. On the other hand, only having to consider the one-player case thanks to the one-to-two-player lift is of tremendous help in practice. A generalization of the one-to-two-player lift to *stochastic games* was shown also by Gimbert and Zielonka in an unpublished paper [34] and is about memoryless strategies that are *pure* (i.e., not using randomization).

The need for memory. Recent research tends to study increasingly complex settings – such as combinations of qualitative/quantitative objectives or of behavioral models – for which finite or infinite memory is often required; see examples in deterministic games [17, 50, 12, 8, 14], Markov decision processes – i.e., one-player stochastic games [46, 47, 24, 3, 11], or stochastic games [28, 18, 25, 23, 40]. Motivated by the growing memory requirements of these endeavors, research about strategy complexity often turns toward *finite-memory determinacy.* Proving finite-memory determinacy is sometimes difficult (already in deterministic games, e.g., [6]), and as opposed to memoryless strategies, there are few widely applicable results. We mention [38], which provides sufficient conditions for finite-memory determinacy in Boolean combinations of finite-memory-determined objectives in deterministic games. Results for multi-player non-zero-sum games are also available [37].

Arena-independent finite-memory. A middle ground between the well-understood memoryless determinacy and the more puzzling finite-memory determinacy was proposed for deterministic games in [7]: an objective is said to admit *arena-independent finite-memory (AIFM) strategies* if a *single finite memory structure* suffices to play optimally in any arena. In practice, this memory structure may depend on parameters of the objective (for instance, largest weight, number of priorities), but not on parameters intrinsically linked to the arena (e.g., number of states or transitions). AIFM strategies include as a special case memoryless strategies, since they can be implemented with a trivial memory structure with a single state.

AIFM strategies have a remarkable feature: in deterministic arenas, AIFM generalizations of *both* characterizations from [33] hold, including the one-to-two-player lift [7]. From a practical point of view, it brings techniques usually linked to memoryless determinacy to many finite-memory-determined objectives. The aim of this article is to show that this also holds true in stochastic arenas.

Contributions. We provide an overview of desirable properties of objectives in which pure AIFM strategies suffice to play optimally in *stochastic* games, and tools to study them. This entails: (a) a proof of a specific feature of objectives for which pure AIFM strategies suffice to play optimally: for such objectives, there also exist pure AIFM *subgame perfect (SP)* strategies (Theorem 7), which is a stronger requirement than optimality; (b) a more general one-to-two-player lift: we show the equivalence between the existence of pure AIFM optimal strategies in two-player games for both players and the existence of pure AIFM optimal strategies in one-player games, thereby simplifying the proof of memory requirements for many objectives (Theorem 8); (c) two intuitive conditions generalizing monotony and selectivity in the stochastic/AIFM case, which are equivalent to the sufficiency of pure AIFM strategies to play optimally in *one-player stochastic arenas* (Theorem 13) for all objectives that can be encoded as *real payoff functions*. In practice, (c) can be used to prove memory requirements in one-player arenas, and then (b) can be used to lift these to the two-player case.

These results reinforce both sides on the frontier between AIFM strategies and general finite-memory strategies: on the one hand, objectives for which pure AIFM strategies suffice indeed share interesting properties with objectives for which pure memoryless strategies suffice, rendering their analysis easier, even in the stochastic case; on the other hand, our novel result about SP strategies does not hold for (arena-*dependent*) finite-memory strategies, and therefore further distinguishes the AIFM case from the finite-memory case.

The one-to-two-player lift for pure AIFM strategies in stochastic games is not surprising, as it holds for pure memoryless strategies in stochastic games [34], and for AIFM strategies in deterministic games [7]. However, although the monotony/selectivity characterization is inspired from the deterministic case [33, 7], it had not been formulated for stochastic games, even in the pure memoryless case – its proof involves new technical difficulties to which our improved understanding of subgame perfect strategies comes in handy.

All our results are about the optimality of *pure* AIFM strategies in various settings: they can be applied in an independent way for deterministic games and for stochastic games, and they can also consider optimality under restriction to different classes of strategies (allowing or not the use of randomization and infinite memory).

The proof technique for the one-to-two-player lift shares a similar outline in [33, 34, 7] and in this paper: it relies on an induction on the number of edges in arenas to show the existence of memoryless optimal strategies. This edge-induction technique is frequently used in comparable ways in other works about memoryless determinacy [35, 31, 32, 18]. In the AIFM case, the extra challenge consists of applying such an induction to the right set of arenas in order for a result about memoryless strategies to imply something about AIFM strategies. Work in [7] paved the way to neatly overcome this technical hindrance and we were able to factorize the main argument in Lemma 6.

Applicability. Let us discuss objectives that admit, or not, pure AIFM optimal strategies in stochastic arenas.

- Objectives for which AIFM optimal strategies exist include the aforementioned memoryless-determined objectives [29, 22, 48, 10], as explained earlier. Such objectives could already be studied through the lens of a one-to-two-player lift [34], but our two other main results also apply to these.
- Pure AIFM optimal strategies also exist in lexicographic reachability-safety games [23, Theorem 4]: the memory depends only on the number of targets to visit or avoid, but not on parameters of the arena (number of states or transitions).
- Muller objectives whose probability must be maximized [15] also admit pure AIFM optimal strategies: the number of memory states depends only on the colors and on the Muller condition.
- In general, every ω-regular objective admits pure AIFM optimal strategies, as it can be seen as a parity objective (for which pure memoryless strategies suffice) after taking the product of the game graph with a deterministic parity automaton accepting the objective [42, 20]. This parity automaton can be taken as an arena-independent memory structure. It is therefore possible to use our results to investigate precise memory bounds in stochastic games for multiple ω-regular objectives which have been studied in deterministic games or in one-player stochastic games: generalized parity games [21], lower- and upper-bounded energy games [5], some window objectives [13, 11], weak parity games [49].
- There are objectives for which finite-memory strategies suffice for some player, but with an underlying memory structure depending on parameters of the arena (an example is provided by the *Gain* objective in [40, Theorem 6]). Many objectives also require infinite memory, such as generalized mean-payoff games [18] (both in deterministic and stochastic games) and energy-parity games (only in stochastic games [17, 39]). Our characterizations provide a more complete understanding of why AIFM strategies do not suffice.

Deterministic and stochastic games. There are natural ways to extend classical objectives for deterministic games to a stochastic context: typically, for qualitative objectives, a natural stochastic extension is to maximize the probability to win. Still, in general, memory requirements may increase when switching to the stochastic context. To show that understanding the deterministic case is insufficient to understand the stochastic case, we outline three situations displaying different behaviors.

- As mentioned above, for many classical objectives, memoryless strategies suffice both in deterministic and in stochastic games.
- AIFM strategies may suffice both for deterministic and stochastic games, but with a difference in the size of the required memory structure. One such example is provided by the weak parity objective [49], for which memoryless strategies suffice in deterministic

games, but which requires memory in stochastic games (this was already noticed in [34, Section 4.4]). Yet, it is possible to show that pure AIFM strategies suffice in stochastic games using the results from our paper. This shows that to go from the deterministic to the stochastic case, a "constant" increase in memory may be necessary and sufficient.

- There are also objectives for which memoryless strategies suffice in deterministic games, but even AIFM strategies do not suffice in stochastic games. One such example consists in maximizing the probability to obtain a non-negative discounted sum (which is different from maximizing the expected value of the discounted sum, for which memoryless strategies suffice, as is shown in [48]).

Formal proofs for these last two examples are provided in the full version [9]. These three situations further highlight the significance of establishing results about memory requirements in stochastic games, even for objectives whose deterministic version is well-understood.

Outline. We introduce our framework and notations in Section 2. We discuss AIFM strategies and tools to relate them to memoryless strategies in Section 3, which allows us to prove our result about subgame perfect strategies. The one-to-two-player lift is presented in Section 4, followed by the one-player characterization in Section 5. Due to a lack of space, we choose to focus on Section 5 and only sketch Section 4; the complete proofs and technical details are found in the full version of the article [9].

2 Preliminaries

Let C be an arbitrary set of *colors*.

Arenas. For a measurable space $(\Omega, \mathcal{F})$ (resp. a finite set Ω), we write $\mathsf{Dist}(\Omega, \mathcal{F})$ (resp. $\mathsf{Dist}(\Omega)$) for the set of *probability distributions on* $(\Omega, \mathcal{F})$ (resp. *on* Ω). For Ω a finite set and $\mu \in \mathsf{Dist}(\Omega)$, we write $\mathsf{Supp}(\mu) = \{\omega \in \Omega \mid \mu(\omega) > 0\}$ for the *support of* μ.

We consider stochastic games played by two players, called $\mathcal{P}_1$ (for player 1) and $\mathcal{P}_2$ (for player 2), who play in a turn-based fashion on *arenas*. A (two-player stochastic turn-based) *arena* is a tuple $\mathcal{A} = (S_1, S_2, A, \delta, \mathsf{col})$, where: S_1 and S_2 are two disjoint finite sets of *states*, respectively controlled by $\mathcal{P}_1$ and $\mathcal{P}_2$ – we denote $S = S_1 \uplus S_2$; A is a finite set of *actions*; $\delta\colon S \times A \to \mathsf{Dist}(S)$ is a partial function called *probabilistic transition function*; $\mathsf{col}\colon S \times A \to C$ is a partial function called *coloring function*. For a state $s \in S$, we write $A(s)$ for the set of actions that are *available in* s, that is, the set of actions for which $\delta(s, a)$ is defined. For $s \in S$, function col must be defined for all pairs (s, a) such that a is available in s. We require that for all $s \in S$, $A(s) \neq \emptyset$ (i.e., arenas are *non-blocking*).

For $s, s' \in S$ and $a \in A(s)$, we denote $\delta(s, a, s')$ instead of $\delta(s, a)(s')$ for the probability to reach s' in one step by playing a in s, and we write $(s, a, s') \in \delta$ if and only if $\delta(s, a, s') > 0$. An interesting subclass of (stochastic) arenas is the class of *deterministic* arenas: an arena $\mathcal{A} = (S_1, S_2, A, \delta, \mathsf{col})$ is *deterministic* if for all $s \in S$, $a \in A(s)$, $|\mathsf{Supp}(\delta(s, a))| = 1$.

A *play of* $\mathcal{A}$ is an infinite sequence of states and actions $s_0 a_1 s_1 a_2 s_2 \ldots \in (SA)^\omega$ such that for all $i \geq 0$, $(s_i, a_{i+1}, s_{i+1}) \in \delta$. A prefix of a play is an element in $S(AS)^*$ and is called a *history*; the set of all histories starting in a state $s \in S$ is denoted $\mathsf{Hists}(\mathcal{A}, s)$. For $S' \subseteq S$, we write $\mathsf{Hists}(\mathcal{A}, S')$ for the unions of $\mathsf{Hists}(\mathcal{A}, s)$ over all states $s \in S'$. For $\rho = s_0 a_1 s_1 \ldots a_n s_n$ a history, we write $\mathsf{out}(\rho)$ for s_n. For $i \in \{1, 2\}$, we write $\mathsf{Hists}_i(\mathcal{A}, s)$ and $\mathsf{Hists}_i(\mathcal{A}, S')$ for the corresponding histories ρ such that $\mathsf{out}(\rho) \in S_i$. For $s, s' \in S$, we write $\mathsf{Hists}(\mathcal{A}, s, s')$ for the histories in $\mathsf{Hists}(\mathcal{A}, s)$ such that $\mathsf{out}(\rho) = s'$.

We write $\widehat{\mathsf{col}}$ for the extension of col to histories and plays: more precisely, for a history $\rho = s_0a_1s_1 \dots a_ns_n$, $\widehat{\mathsf{col}}(\rho)$ is the finite sequence $\mathsf{col}(s_0, a_1) \dots \mathsf{col}(s_{n-1}, a_n) \in C^*$; for $\pi = s_0a_1s_1a_2s_2 \dots$ a play, $\widehat{\mathsf{col}}(\pi)$ is the infinite sequence $\mathsf{col}(s_0, a_1)\mathsf{col}(s_1, a_2) \dots \in C^\omega$.

A *one-player arena of* $\mathcal{P}_i$ is an arena $\mathcal{A} = (S_1, S_2, A, \delta, \mathsf{col})$ such that for all $s \in S_{3-i}$, $|A(s)| = 1$. A one-player arena corresponds to a *Markov decision process (MDP)* [44, 2].

An *initialized arena* is a pair $(\mathcal{A}, S_{\mathsf{init}})$ such that $\mathcal{A}$ is an arena and S_{init} is a non-empty subset of the states of $\mathcal{A}$, called the set of *initial states.* We assume w.l.o.g. that all states of $\mathcal{A}$ are reachable from S_{init} following transitions with positive probabilities in the probabilistic transition function of $\mathcal{A}$. In case of a single initial state $s \in S$, we write $(\mathcal{A}, s)$ for $(\mathcal{A}, \{s\})$.

We will consider *classes* (sets) of initialized arenas, which are usually denoted by the letter $\mathfrak{A}$. Typical classes that we will consider consist of all one-player or two-player, deterministic or stochastic initialized arenas. We use *initialized* arenas throughout the paper for technical reasons, but our results can be converted to results using the classical notion of arena.

Memory. We define a notion of memory based on complete deterministic automata on *colors.* The goal of using colors instead of states/actions for transitions of the memory is to define memory structures *independently of arenas.* A *memory skeleton* is a tuple $\mathcal{M} = (M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ where M is a set of *memory states*, $m_{\mathsf{init}} \in M$ is an *initial state* and $\alpha_{\mathsf{upd}} \colon M \times C \to M$ is an *update function.* We add the following constraint: for all finite sets of colors $B \subseteq C$, the number of states reachable from m_{init} with transitions provided by $\alpha_{\mathsf{upd}}|_{M\times B}$ is finite (where $\alpha_{\mathsf{upd}}|_{M\times B}$ is the restriction of the domain of α_{upd} to $M \times B$).

Memory skeletons with a finite state space are all encompassed by this definition, but this also allows some skeletons with infinitely many states. For example, if $C = \mathbb{N}$, the tuple $(\mathbb{N}, 0, (m, n) \mapsto \max\{m, n\})$, which remembers the largest color seen, is a valid memory skeleton: for any finite $B \subseteq C$, we only need to use memory states up to $\max B$. However, the tuple $(\mathbb{N}, 0, (m, n) \mapsto m + n)$ remembering the current sum of colors seen is *not* a memory skeleton, as infinitely many states are reachable from 0, even if only $B = \{1\}$ can be used. We denote $\widehat{\alpha_{\mathsf{upd}}} \colon M \times C^* \to M$ for the natural extension of α_{upd} to finite sequences of colors.

Let $\mathcal{M}^1 = (M^1, m^1_{\mathsf{init}}, \alpha^1_{\mathsf{upd}})$ and $\mathcal{M}^2 = (M^2, m^2_{\mathsf{init}}, \alpha^2_{\mathsf{upd}})$ be memory skeletons. Their *product* $\mathcal{M}^1 \otimes \mathcal{M}^2$ is the memory skeleton $(M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ with $M = M^1 \times M^2$, $m_{\mathsf{init}} = (m^1_{\mathsf{init}}, m^2_{\mathsf{init}})$, and, for all $m^1 \in M^1$, $m^2 \in M^2$, $c \in C$, $\alpha_{\mathsf{upd}}((m^1, m^2), c) = (\alpha^1_{\mathsf{upd}}(m^1, c), \alpha^2_{\mathsf{upd}}(m^2, c))$. The update function of the product simply updates both skeletons in parallel.

Strategies. Given an initialized arena $(\mathcal{A} = (S_1, S_2, A, \delta, \mathsf{col}), S_{\mathsf{init}})$ and $i \in \{1, 2\}$, a *strategy of* $\mathcal{P}_i$ *on* $(\mathcal{A}, S_{\mathsf{init}})$ is a function $\sigma_i \colon \mathsf{Hists}_i(\mathcal{A}, S_{\mathsf{init}}) \to \mathsf{Dist}(A)$ such that for all $\rho \in \mathsf{Hists}_i(\mathcal{A}, S_{\mathsf{init}})$, $\mathsf{Supp}(\sigma_i(\rho)) \subseteq A(\mathsf{out}(\rho))$. For $i \in \{1, 2\}$, we denote by $\Sigma^{\mathsf{G}}_i(\mathcal{A}, S_{\mathsf{init}})$ the set of all strategies of $\mathcal{P}_i$ on $(\mathcal{A}, S_{\mathsf{init}})$.

A strategy σ_i of $\mathcal{P}_i$ on $(\mathcal{A}, S_{\mathsf{init}})$ is *pure* if for all $\rho \in \mathsf{Hists}_i(\mathcal{A}, S_{\mathsf{init}})$, $|\mathsf{Supp}(\sigma_i(\rho))| = 1$. If a strategy is not pure, then it is *randomized.* A strategy σ_i of $\mathcal{P}_i$ on $(\mathcal{A}, S_{\mathsf{init}})$ is *memoryless* if for all $\rho, \rho' \in \mathsf{Hists}_i(\mathcal{A}, S_{\mathsf{init}})$, $\mathsf{out}(\rho) = \mathsf{out}(\rho')$ implies $\sigma_i(\rho) = \sigma_i(\rho')$. A *pure memoryless* strategy of $\mathcal{P}_i$ can be simply specified as a function $S_i \to A$. A strategy σ_i of $\mathcal{P}_i$ on $(\mathcal{A}, S_{\mathsf{init}})$ is *finite-memory* if it can be encoded as a *Mealy machine* $\Gamma = (\mathcal{M}, \alpha_{\mathsf{nxt}})$, with $\mathcal{M} = (M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ a memory skeleton and $\alpha_{\mathsf{nxt}} \colon S_i \times M \to \mathsf{Dist}(A)$ being the *next-action function*, which is such that for $s \in S_i$, $m \in M$, $\mathsf{Supp}(\alpha_{\mathsf{nxt}}(s, m)) \subseteq A(s)$. Strategy σ_i is encoded by Γ if for all histories $\rho \in \mathsf{Hists}_i(\mathcal{A}, S_{\mathsf{init}})$, $\sigma_i(\rho) = \alpha_{\mathsf{nxt}}(\mathsf{out}(\rho), \widehat{\alpha_{\mathsf{upd}}}(m_{\mathsf{init}}, \widehat{\mathsf{col}}(\rho)))$. If σ_i can be encoded as a Mealy machine $(\mathcal{M}, \alpha_{\mathsf{nxt}})$, we say that σ_i is *based on (memory)* $\mathcal{M}$. If σ_i is *based on* $\mathcal{M}$ and is *pure*, then the next-action function can be specified as a function $S_i \times M \to A$. Memoryless strategies correspond to finite-memory strategies based on the *trivial memory skeleton* $\mathcal{M}_{\mathsf{triv}} = (\{m_{\mathsf{init}}\}, m_{\mathsf{init}}, (m_{\mathsf{init}}, c) \mapsto m_{\mathsf{init}})$ that has a single state.

We denote by $\Sigma_i^{\mathsf{PFM}}(\mathcal{A}, S_{\mathsf{init}})$ (resp. $\Sigma_i^{\mathsf{P}}(\mathcal{A}, S_{\mathsf{init}})$, $\Sigma_i^{\mathsf{GFM}}(\mathcal{A}, S_{\mathsf{init}})$, $\Sigma_i^{\mathsf{G}}(\mathcal{A}, S_{\mathsf{init}})$) the set of pure finite-memory (resp. pure, finite-memory, general) strategies of $\mathcal{P}_i$ on $(\mathcal{A}, S_{\mathsf{init}})$. A *type of strategies* is an element $\mathsf{X} \in \{\mathsf{PFM}, \mathsf{P}, \mathsf{GFM}, \mathsf{G}\}$ corresponding to these subsets.

Outcomes. Let $(\mathcal{A} = (S_1, S_2, A, \delta, \mathsf{col}), S_{\mathsf{init}})$ be an initialized arena. When both players have decided on a strategy and an initial state has been chosen, the generated object is a (finite or countably infinite) Markov chain, which induces a probability distribution on the plays. For strategies σ_1 of $\mathcal{P}_1$ and σ_2 of $\mathcal{P}_2$ on $(\mathcal{A}, S_{\mathsf{init}})$ and $s \in S_{\mathsf{init}}$, we denote $\mathsf{P}_{\mathcal{A},s}^{\sigma_1,\sigma_2}$ for the probability distribution on plays induced by σ_1 and σ_2, starting from state s.

We define $\mathcal{F}$ to be the smallest σ-algebra on C^ω generated by the set of all *cylinders* on C. In particular, every probability distribution $\mathsf{P}_{\mathcal{A},s}^{\sigma_1,\sigma_2}$ naturally induces a probability distribution over $(C^\omega, \mathcal{F})$ through the $\widehat{\mathsf{col}}$ function, which we denote $\mathsf{Pc}_{\mathcal{A},s}^{\sigma_1,\sigma_2}$.

Preferences. To specify each player's objective, we use the general notion of *preference relation*. A *preference relation* $\sqsubseteq$ *(on C)* is a total preorder over $\mathsf{Dist}(C^\omega, \mathcal{F})$. The idea is that $\mathcal{P}_1$ favors the distributions in $\mathsf{Dist}(C^\omega, \mathcal{F})$ that are the largest for $\sqsubseteq$, and as we are studying *zero-sum* games, $\mathcal{P}_2$ favors the distributions that are the smallest for $\sqsubseteq$. For $\sqsubseteq$ a preference relation and $\mu, \mu' \in \mathsf{Dist}(C^\omega, \mathcal{F})$, we write $\mu \sqsubset \mu'$ if $\mu \sqsubseteq \mu'$ and $\mu' \not\sqsubseteq \mu$.

Depending on the context, it might not be necessary to define a preference relation as *total*: it is sufficient to order distributions that can arise as an element $\mathsf{P}_{\mathcal{A},s}^{\sigma_1,\sigma_2}$. For example, in the specific case of deterministic games in which only pure strategies are considered, all distributions that arise are always Dirac distributions on a single infinite word in C^ω. In this context, it is therefore sufficient to define a total preorder over all Dirac distributions (which we can then see as infinite words, giving a definition of preference relation similar to [33, 7]). We give some examples to illustrate our notion of *preference relation*.

▶ **Example 1.** We give three examples corresponding to three different ways to encode preference relations. First, a preference relation can be induced by an event $W \in \mathcal{F}$ called a *winning condition*, which consists of infinite sequences of colors. The objective of $\mathcal{P}_1$ is to maximize the probability that the event W happens. An event W naturally induces a preference relation $\sqsubseteq_W$ such that for $\mu, \mu' \in \mathsf{Dist}(C^\omega, \mathcal{F})$, $\mu \sqsubseteq_W \mu'$ if and only if $\mu(W) \le \mu'(W)$. For $C = \mathbb{N}$, we give the example of the *weak parity* winning condition W_{wp} [49], defined as $W_{\mathsf{wp}} = \{c_1 c_2 \ldots \in C^\omega \mid \max_{j \ge 1} c_j \text{ exists and is even}\}$. In finite arenas, the value $\max_{j \ge 1} c_j$ always exists, as there are only finitely many colors that appear. This is different from the classical parity condition, which requires the maximal color *seen infinitely often* to be even, and not just the maximal color seen.

A preference relation can also be induced by a Borel *(real) payoff function* $f \colon C^\omega \to \mathbb{R}$. For example, if $C = \mathbb{R}$ and $\lambda \in (0, 1)$, a classical payoff function [48] is the *discounted sum* Disc_λ, defined for $c_1 c_2 \ldots \in C^\omega$ as $\mathsf{Disc}_\lambda(c_1 c_2 \ldots) = \lim_n \sum_{i=0}^n \lambda^i \cdot c_{i+1}$. The goal of $\mathcal{P}_1$ is to maximize the expected value of f, which is defined for a probability distribution $\mu \in \mathsf{Dist}(C^\omega, \mathcal{F})$ as $\mathsf{E}_\mu[f] = \int f \,\mathrm{d}\mu$. A payoff function f naturally induces a preference relation $\sqsubseteq_f$: for $\mu_1, \mu_2 \in \mathsf{Dist}(C^\omega, \mathcal{F})$, $\mu_1 \sqsubseteq_f \mu_2$ if and only if $\mathsf{E}_{\mu_1}[f] \le \mathsf{E}_{\mu_2}[f]$. Payoff functions are more general than winning conditions: for W a winning condition, the preference relation induced by the indicator function of W corresponds to the preference relation induced by W.

It is also possible to specify preference relations that cannot be expressed as a payoff function. An example is given in [27]: we assume that the goal of $\mathcal{P}_1$ is to see color $c \in C$ with probability *precisely* $\frac{1}{2}$. We denote the event of seeing color c as $\Diamond c \in \mathcal{F}$. Then for $\mu, \mu' \in \mathsf{Dist}(C^\omega, \mathcal{F})$, $\mu \sqsubseteq \mu'$ if and only if $\mu(\Diamond c) \neq \frac{1}{2}$ or $\mu'(\Diamond c) = \frac{1}{2}$. ⌟

A (two-player stochastic turn-based zero-sum) *initialized game* is a tuple $\mathcal{G} = (\mathcal{A}, S_{\mathsf{init}}, \sqsubseteq)$, where $(\mathcal{A}, S_{\mathsf{init}})$ is an initialized arena and $\sqsubseteq$ is a preference relation.

Optimality. Let $\mathcal{G} = (\mathcal{A}, S_{\mathsf{init}}, \sqsubseteq)$ be an initialized game and $\mathsf{X} \in \{\mathsf{PFM}, \mathsf{P}, \mathsf{GFM}, \mathsf{G}\}$ be a type of strategies. For $s \in S_{\mathsf{init}}$, $\sigma_1 \in \Sigma_1^{\mathsf{X}}(\mathcal{A}, S_{\mathsf{init}})$, we define

$$\mathsf{UCol}_{\sqsubseteq}^{\mathsf{X}}(\mathcal{A}, s, \sigma_1) = \{\mu \in \mathsf{Dist}(C^\omega, \mathcal{F}) \mid \exists\, \sigma_2 \in \Sigma_2^{\mathsf{X}}(\mathcal{A}, s), \mathsf{Pc}_{\mathcal{A},s}^{\sigma_1,\sigma_2} \sqsubseteq \mu\}.$$

The set $\mathsf{UCol}_{\sqsubseteq}^{\mathsf{X}}(\mathcal{A}, s, \sigma_1)$ corresponds to all the distributions that are at least as good for $\mathcal{P}_1$ (w.r.t. $\sqsubseteq$) as a distribution that $\mathcal{P}_2$ can induce by playing a strategy σ_2 of type X against σ_1; this set is upward-closed w.r.t. $\sqsubseteq$. For $\sigma_1, \sigma_1' \in \Sigma_1^{\mathsf{X}}(\mathcal{A}, S_{\mathsf{init}})$, we say that σ_1 is *at least as good as σ_1' from $s \in S_{\mathsf{init}}$ under X strategies* if $\mathsf{UCol}_{\sqsubseteq}^{\mathsf{X}}(\mathcal{A}, s, \sigma_1) \subseteq \mathsf{UCol}_{\sqsubseteq}^{\mathsf{X}}(\mathcal{A}, s, \sigma_1')$. This inclusion means that the best replies of $\mathcal{P}_2$ against σ_1' yield an outcome that is at least as bad for $\mathcal{P}_1$ (w.r.t. $\sqsubseteq$) as the best replies of $\mathcal{P}_2$ against σ_1. We can define symmetrical notions for strategies of $\mathcal{P}_2$.

Let $\mathcal{G} = (\mathcal{A}, S_{\mathsf{init}}, \sqsubseteq)$ be an initialized game and $\mathsf{X} \in \{\mathsf{PFM}, \mathsf{P}, \mathsf{GFM}, \mathsf{G}\}$ be a type of strategies. A strategy $\sigma_i \in \Sigma_i^{\mathsf{X}}(\mathcal{A}, S_{\mathsf{init}})$ is X*-optimal in $\mathcal{G}$* if it is at least as good under X strategies as any other strategy in $\Sigma_i^{\mathsf{X}}(\mathcal{A}, S_{\mathsf{init}})$ from all $s \in S_{\mathsf{init}}$.

When the considered preference relation $\sqsubseteq$ is clear, we often talk about X-optimality in an initialized arena $(\mathcal{A}, S_{\mathsf{init}})$ to refer to X-optimality in the initialized game $(\mathcal{A}, S_{\mathsf{init}}, \sqsubseteq)$. Given a preference relation, a class of arenas, and a type of strategies, our goal is to understand what kinds of strategies are sufficient to play optimally. In the following definition, abbreviations *AIFM* and *FM* stand respectively for *arena-independent finite-memory* and *finite-memory.*

▶ **Definition 2.** *Let $\sqsubseteq$ be a preference relation, $\mathfrak{A}$ be a class of initialized arenas, $\mathsf{X} \in \{\mathsf{PFM}, \mathsf{P}, \mathsf{GFM}, \mathsf{G}\}$ be a type of strategies, and $\mathcal{M}$ be a memory skeleton. We say that* pure AIFM strategies suffice to play X-optimally in $\mathfrak{A}$ for $\mathcal{P}_1$ *if there exists a memory skeleton $\mathcal{M}$ such that for all $(\mathcal{A}, S_{\mathsf{init}}) \in \mathfrak{A}$, $\mathcal{P}_1$ has a pure strategy based on $\mathcal{M}$ that is X-optimal in $(\mathcal{A}, S_{\mathsf{init}})$. We say that* pure FM strategies suffice to play X-optimally in $\mathfrak{A}$ for $\mathcal{P}_1$ *if for all $(\mathcal{A}, S_{\mathsf{init}}) \in \mathfrak{A}$, there exists a memory skeleton $\mathcal{M}$ such that $\mathcal{P}_1$ has a pure strategy based on $\mathcal{M}$ that is X-optimal in $(\mathcal{A}, S_{\mathsf{init}})$.*

Since memoryless strategies are a specific kind of finite-memory strategies based on the same memory skeleton $\mathcal{M}_{\mathsf{triv}}$, the sufficiency of pure memoryless strategies is equivalent to the sufficiency of pure strategies based on $\mathcal{M}_{\mathsf{triv}}$, and is therefore a specific case of the sufficiency of pure AIFM strategies. Notice the difference between the order of quantifiers for AIFM and FM strategies: the sufficiency of pure AIFM strategies implies the sufficiency of pure FM strategies, but the opposite is false in general (an example is given in [17]).

▶ **Example 3.** Let us reconsider the weak parity winning condition W_{wp} introduced in Example 1: the goal of $\mathcal{P}_1$ is to maximize the probability that the greatest color seen is even. To play optimally in any stochastic game, it is sufficient for both players to remember the largest color already seen, which can be implemented by the memory skeleton $\mathcal{M}_{\mathsf{max}} = (\mathbb{N}, 0, (m, n) \mapsto \max\{m, n\})$. As explained above, this memory skeleton has an infinite state space, but as there are only finitely many colors in every (finite) arena, only a finite part of the skeleton is sufficient to play optimally in any given arena. The size of the skeleton used for a fixed arena depends on the appearing colors, but for a fixed number of colors, it does not depend on parameters of the arena (such as its state and action spaces). Therefore pure AIFM strategies suffice to play optimally for both players, and more precisely pure strategies based on $\mathcal{M}_{\mathsf{max}}$ suffice for both players. ⌟

We define a second stronger notion related to optimality of strategies, which is the notion of *subgame perfect strategy*: a strategy is *subgame perfect* in a game if it reacts optimally to all histories consistent with the arena, even histories not consistent with the strategy itself, or histories that only a non-rational adversary would play [43]. This is a desirable property of strategies that is stronger than optimality, since a subgame perfect strategy is not only optimal from the initial position, but from any arbitrary stage (*subgame*) of the game. In particular, if an opponent plays non-optimally, an optimal strategy that is not subgame perfect does not always fully exploit the advantage that the opponent's suboptimal behavior provides, and may yield a result that is not optimal *when starting in a subgame*. We first need extra definitions.

For $w \in C^*$, $\mu \in \mathsf{Dist}(C^\omega, \mathcal{F})$, we define the *shifted distribution* $w\mu$ as the distribution such that for an event $E \in \mathcal{F}$, $w\mu(E) = \mu(\{w' \in C^\omega \mid ww' \in E\})$.

For $(\mathcal{A}, S_{\mathsf{init}})$ an initialized arena, for $\sigma_i \in \Sigma_i^{\mathsf{G}}(\mathcal{A}, S_{\mathsf{init}})$, and for $\rho = s_0a_1s_1 \dots a_ns_n \in \mathsf{Hists}(\mathcal{A}, S_{\mathsf{init}})$, we define the *shifted strategy* $\sigma_i[\rho] \in \Sigma_i^{\mathsf{G}}(\mathcal{A}, \mathsf{out}(\rho))$ as the strategy such that, for $\rho' = s_na_{n+1}s_{n+1} \dots a_ms_m \in \mathsf{Hists}_i(\mathcal{A}, \mathsf{out}(\rho))$, $\sigma_i[\rho](\rho') = \sigma_i(s_0a_1s_1 \dots a_ms_m)$.

For $\sqsubseteq$ a preference relation and $w \in C^*$, we define the *shifted preference relation* $\sqsubseteq_{[w]}$ as the preference relation such that for $\mu, \mu' \in \mathsf{Dist}(C^\omega, \mathcal{F})$, $\mu \sqsubseteq_{[w]} \mu'$ if and only if $w\mu \sqsubseteq w\mu'$.

▶ **Definition 4.** *Let $\mathcal{G} = (\mathcal{A}, S_{\mathsf{init}}, \sqsubseteq)$ be an initialized game and $\mathsf{X} \in \{\mathsf{PFM}, \mathsf{P}, \mathsf{GFM}, \mathsf{G}\}$ be a type of strategies. A strategy $\sigma_i \in \Sigma_i^{\mathsf{X}}(\mathcal{A}, S_{\mathsf{init}})$ is* X-subgame perfect (X-SP) in *$\mathcal{G}$ if for all $\rho \in \mathsf{Hists}(\mathcal{A}, S_{\mathsf{init}})$, shifted strategy $\sigma_i[\rho]$ is X-optimal in the initialized game $(\mathcal{A}, \mathsf{out}(\rho), \sqsubseteq_{[\widehat{\mathsf{col}}(\rho)]})$.*

Strategies that are X-SP are in particular X-optimal; the converse is not true in general.

3 Coverability and subgame perfect strategies

In this section, we establish a key tool (Lemma 6) which can be used to reduce questions about the sufficiency of *AIFM strategies* in reasonable classes of initialized arenas to the sufficiency of *memoryless strategies* in a subclass. We then describe the use of this lemma to obtain our first main result (Theorem 7), which shows that the sufficiency of pure AIFM strategies implies the existence of pure AIFM *SP* strategies. Technical details are in [9].

▶ **Definition 5.** *An initialized arena $((S_1, S_2, A, \delta, \mathsf{col}), S_{\mathsf{init}})$* is covered by memory skeleton *$\mathcal{M} = (M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ if there exists a function $\phi\colon S \to M$ such that for all $s \in S_{\mathsf{init}}$, $\phi(s) = m_{\mathsf{init}}$, and for all $(s, a, s') \in \delta$, $\alpha_{\mathsf{upd}}(\phi(s), \mathsf{col}(s, a)) = \phi(s')$.*

The *coverability* property means that it is possible to assign a unique memory state to each arena state such that transitions of the arena always update the memory state in a way that is consistent with the memory skeleton. A covered initialized arena carries in some way already sufficient information to play with memory $\mathcal{M}$ without actually using memory – using the memory skeleton $\mathcal{M}$ would not be more powerful than using no memory at all. This property is linked to the classical notion of *product arena with* $\mathcal{M}$: a strategy based on $\mathcal{M}$ corresponds to a memoryless strategy in a product arena (e.g., [7, Lemma 1] and [9]). For our results, *coverability* is a key technical definition, as the class of initialized arenas covered by a memory skeleton is sufficiently well-behaved to support edge-induction arguments, whereas it is difficult to perform such techniques directly on the class of *product arenas*: removing a single edge from a product arena makes it hard to express as a product arena, whereas it is clear that coverability is preserved. Every initialized arena is covered by $\mathcal{M}_{\mathsf{triv}}$, which is witnessed by the function ϕ associating m_{init} to every state.

Definitions close to our notion of *coverability by* $\mathcal{M}$ were introduced for deterministic arenas in [36, 7]. The definition of *adherence with* $\mathcal{M}$ in [36, Definition 8.12] is very similar, but does not distinguish initial states from the rest (neither in the arena nor in the memory skeleton). Our property of $(\mathcal{A}, S_{\mathsf{init}})$ being *covered by* $\mathcal{M}$ is also equivalent to $\mathcal{A}$ being both *prefix-covered* and *cyclic-covered* by $\mathcal{M}$ from S_{init} [7]. Distinguishing both notions gives insight in [7] as they are used at different places in proofs (*prefix-covered* along with *monotony*, and *cyclic-covered* along with *selectivity*). Here, we opt for a single concise definition.

The following lemma sums up our practical use of the idea of *coverability*.

▶ **Lemma 6.** *Let* $\sqsubseteq$ *be a preference relation,* $\mathcal{M}$ *be a memory skeleton, and* $\mathsf{X} \in \{\mathsf{PFM}, \mathsf{P}, \mathsf{GFM}, \mathsf{G}\}$ *be a type of strategies. Let* $\mathfrak{A}$ *be the class of one-player or two-player, stochastic or deterministic initialized arenas. Then,* $\mathcal{P}_1$ *has an* X*-optimal (resp.* X*-SP) strategy based on* $\mathcal{M}$ *in all initialized arenas in* $\mathfrak{A}$ *if and only if* $\mathcal{P}_1$ *has a memoryless* X*-optimal (resp.* X*-SP) strategy in all initialized arenas covered by* $\mathcal{M}$ *in* $\mathfrak{A}$*.*

We now state one of our main results, which shows that the sufficiency of pure strategies based on the same memory skeleton $\mathcal{M}$ implies that pure *SP* strategies based on $\mathcal{M}$ exist.

▶ **Theorem 7.** *Let* $\sqsubseteq$ *be a preference relation,* $\mathcal{M}$ *be a memory skeleton, and* $\mathsf{X} \in \{\mathsf{PFM}, \mathsf{P}, \mathsf{GFM}, \mathsf{G}\}$ *be a type of strategies. Let* $\mathfrak{A}$ *be the class of one-player or two-player, stochastic or deterministic initialized arenas. If* $\mathcal{P}_1$ *has pure* X*-optimal strategies based on* $\mathcal{M}$ *in initialized arenas of* $\mathfrak{A}$*, then* $\mathcal{P}_1$ *has pure* X*-SP strategies based on* $\mathcal{M}$ *in initialized arenas of* $\mathfrak{A}$*.*

Proof sketch. By Lemma 6, we can prove instead that $\mathcal{P}_1$ has a pure memoryless X-SP strategy in every initialized arena covered by $\mathcal{M}$, based on the hypothesis that $\mathcal{P}_1$ has a pure memoryless X-optimal strategy in every initialized arena covered by $\mathcal{M}$. For $(\mathcal{A}, S_{\mathsf{init}}) \in \mathfrak{A}$ covered by $\mathcal{M}$, $\mathcal{P}_1$ has a pure memoryless X-optimal strategy σ_1^0. If this strategy is not X-SP, there must be a prefix $\rho \in \mathsf{Hists}(\mathcal{A}, S_{\mathsf{init}})$ such that σ_1^0 is not X-optimal in $(\mathcal{A}, \mathsf{out}(\rho), \sqsubseteq_{[\widehat{\mathsf{col}}(\rho)]})$. Then, we extend $(\mathcal{A}, S_{\mathsf{init}})$ by adding a "chain" of states with colors $\widehat{\mathsf{col}}(\rho)$ up to $\mathsf{out}(\rho)$, and add as an initial state the first state of this chain. *This new arena is still covered by* $\mathcal{M}$, thus $\mathcal{P}_1$ has a pure memoryless X-optimal strategy in this arena that is now X-optimal after seeing ρ. If this strategy is not X-SP, we keep iterating our reasoning. This iteration necessarily ends, as we consider *finite* arenas, on which there are *finitely many pure memoryless strategies.* ◀

This result shows a major distinction between the sufficiency of AIFM strategies and the more general sufficiency of FM strategies: if a player can always play optimally with the same memory, then SP strategies may be played with the same memory as optimal strategies – if a player can play optimally but needs arena-*dependent* finite memory, then infinite memory may still be required to obtain SP strategies. One such example is provided in [38, Example 16] for the *average-energy games with lower-bounded energy* in deterministic arenas: $\mathcal{P}_1$ can always play optimally with pure finite-memory strategies [6, Theorem 13], but infinite memory is needed for SP strategies. As will be further explained later, we will also use Theorem 7 to gain technical insight in the proof of the main result of Section 5.

4 One-to-two-player lift

Our goal in this section is to expose a practical tool to help study the memory requirements of two-player stochastic (or deterministic) games. This tool consists in reducing the study of the sufficiency of pure AIFM strategies for both players in *two*-player games to *one*-player games. We first state our result, and we then explain how it relates to similar results from the literature and sketch its proof. A slightly generalized result with a more fine-grained quantification on the classes of initialized arenas is in [9], with the complete proof.

▶ **Theorem 8** (Pure AIFM one-to-two-player lift). *Let $\sqsubseteq$ be a preference relation, $\mathcal{M}_1$ and $\mathcal{M}_2$ be two memory skeletons, and $\mathsf{X} \in \{\mathsf{PFM}, \mathsf{P}, \mathsf{GFM}, \mathsf{G}\}$ be a type of strategies. Let $\mathfrak{A}$ be the class of all initialized stochastic or deterministic arenas.*

Assume that in all initialized one-player arenas of $\mathcal{P}_1$ in $\mathfrak{A}$, $\mathcal{P}_1$ can play X-optimally with a pure strategy based on $\mathcal{M}_1$, and in all initialized one-player arenas of $\mathcal{P}_2$ in $\mathfrak{A}$, $\mathcal{P}_2$ can play X-optimally with a pure strategy based on $\mathcal{M}_2$. Then in all initialized two-player arenas in $\mathfrak{A}$, both players have a pure X-SP strategy based on $\mathcal{M}_1 \otimes \mathcal{M}_2$.

The practical usage of this result can be summed up as follows: to determine whether pure AIFM strategies are sufficient for both players in stochastic (resp. deterministic) arenas to play X-optimally, it is sufficient to prove it for stochastic (resp. deterministic) *one-player* arenas. Our theorem deals in a uniform manner with stochastic and deterministic arenas, under different types of strategies. Studying memory requirements of one-player arenas is significantly easier than doing so in two-player arenas, as a one-player arena can be seen as a graph (in the deterministic case) or an MDP (in the stochastic case). Still, we bring more tools to study memory requirements of one-player arenas in Section 5.

Theorem 8 generalizes known one-to-two-player lifts: for pure memoryless strategies in deterministic [33] and stochastic [34] games, and for pure AIFM strategies in deterministic games [7]. Very briefly, our proof technique consists in extending the lift for *pure memoryless strategies* in stochastic games [34] in order to deal with *initialized* arenas. Then, we show that this pure memoryless one-to-two-player lift can be applied to the class of *initialized arenas covered by* $\mathcal{M}_1 \otimes \mathcal{M}_2$ (using an edge-induction technique), and Lemma 6 permits to go back from pure memoryless strategies to pure strategies based on $\mathcal{M}_1 \otimes \mathcal{M}_2$. Thanks to Theorem 7, we also go further in our understanding of the optimal strategies: we obtain the existence of X-*SP* strategies instead of the seemingly weaker existence of X-optimal strategies.

5 AIFM characterization

For this section, we fix $\sqsubseteq$ a preference relation, $\mathsf{X} \in \{\mathsf{PFM}, \mathsf{P}, \mathsf{GFM}, \mathsf{G}\}$ a type of strategies, and $\mathcal{M} = (M, m_{\mathsf{init}}, \alpha_{\mathsf{upd}})$ a memory skeleton. We distinguish two classes of initialized arenas: the class $\mathfrak{A}^{\mathsf{D}}_{\mathcal{P}_1}$ of all initialized one-player deterministic arenas of $\mathcal{P}_1$, and the class $\mathfrak{A}^{\mathsf{S}}_{\mathcal{P}_1}$ of all initialized one-player stochastic arenas of $\mathcal{P}_1$. A class of arenas will therefore be specified by a letter $\mathsf{Y} \in \{\mathsf{D}, \mathsf{S}\}$, which we fix for the whole section. Our aim is to give a better understanding of the preference relations for which pure strategies based on $\mathcal{M}$ suffice to play X-optimally in $\mathfrak{A}^{\mathsf{Y}}_{\mathcal{P}_1}$, by *characterizing* it through two intuitive conditions. All definitions and proofs are stated from the point of view of $\mathcal{P}_1$. As we only work with one-player arenas in this section, we abusively write $\mathsf{P}^{\sigma_1}_{\mathcal{A},s}$ and $\mathsf{Pc}^{\sigma_1}_{\mathcal{A},s}$ for the distributions on plays and colors induced by a strategy σ_1 of $\mathcal{P}_1$ on $(\mathcal{A}, s)$, with the unique, trivial strategy for $\mathcal{P}_2$.

For $\mathcal{A} \in \mathfrak{A}^{\mathsf{Y}}_{\mathcal{P}_1}$ and s a state of $\mathcal{A}$, we write $[\mathcal{A}]^{\mathsf{X}}_s = \{\mathsf{Pc}^{\sigma_1}_{\mathcal{A},s} \mid \sigma_1 \in \Sigma^{\mathsf{X}}_1(\mathcal{A}, s)\}$ for the set of all distributions over $(C^\omega, \mathcal{F})$ induced by strategies of type X in $\mathcal{A}$ from s.

For $m_1, m_2 \in M$, we write $L_{m_1,m_2} = \{w \in C^* \mid \widehat{\alpha_{\mathsf{upd}}}(m_1, w) = m_2\}$ for the language of words that are read from m_1 up to m_2 in $\mathcal{M}$. Such a language can be specified by the deterministic automaton that is simply the memory skeleton $\mathcal{M}$ with m_1 as the initial state and m_2 as the unique final state. We extend the *shifted distribution* notation to *sets* of distributions: for $w \in C^*$, for $\Lambda \subseteq \mathsf{Dist}(C^\omega, \mathcal{F})$, we write $w\Lambda$ for the set $\{w\mu \mid \mu \in \Lambda\}$.

Given $\sqsubseteq$ a preference relation, we also extend $\sqsubseteq$ to sets of distributions: for $\Lambda_1, \Lambda_2 \subseteq \mathsf{Dist}(C^\omega, \mathcal{F})$, we write $\Lambda_1 \sqsubseteq \Lambda_2$ if for all $\mu_1 \in \Lambda_1$, there exists $\mu_2 \in \Lambda_2$ such that $\mu_1 \sqsubseteq \mu_2$; we write $\Lambda_1 \sqsubset \Lambda_2$ if there exists $\mu_2 \in \Lambda_2$ such that for all $\mu_1 \in \Lambda_1$, $\mu_1 \sqsubset \mu_2$. Notice that $\neg(\Lambda_1 \sqsubseteq \Lambda_2)$ is equivalent to $\Lambda_2 \sqsubset \Lambda_1$. If Λ_1 is a singleton $\{\mu_1\}$, we write $\mu_1 \sqsubseteq \Lambda_2$ for

$\{\mu_1\} \sqsubseteq \Lambda_2$ (and similarly for Λ_2, and similarly using $\sqsubset$). Notice that $\mu_1 \sqsubseteq \mu_2$ is equivalent to $\{\mu_1\} \sqsubseteq \{\mu_2\}$, so this notational shortcut is sound. For two initialized arenas $(\mathcal{A}_1, s_1)$ and $(\mathcal{A}_2, s_2)$, the inequality $[\mathcal{A}_1]^{\mathsf{X}}_{s_1} \sqsubseteq [\mathcal{A}_2]^{\mathsf{X}}_{s_2}$ means that for every strategy on $(\mathcal{A}_1, s_1)$, there is a strategy on $(\mathcal{A}_2, s_2)$ that induces a distribution that is at least as good.

For two arenas $\mathcal{A}_1$ and $\mathcal{A}_2$ with disjoint state spaces, if s_1 and s_2 are two states controlled by $\mathcal{P}_1$ that are respectively in $\mathcal{A}_1$ and $\mathcal{A}_2$ with disjoint sets of available actions, we write $(\mathcal{A}_1, s_1) \sqcup (\mathcal{A}_2, s_2)$ for the *merged arena* in which s_1 and s_2 are merged, and everything else is kept the same. The merged state which comes from the merge of s_1 and s_2 is usually called t. Formally, let $\mathcal{A}_1 = (S^1_1, S^1_2, A^1, \delta^1, \mathsf{col}^1)$, $\mathcal{A}_2 = (S^2_1, S^2_2, A^2, \delta^2, \mathsf{col}^2)$, $s_1 \in S^1_1$, and $s_2 \in S^2_1$. We assume that $S^1 \cap S^2 = \emptyset$ and that $A(s_1) \cap A(s_2) = \emptyset$. We define $(\mathcal{A}_1, s_1) \sqcup (\mathcal{A}_2, s_2)$ as the arena $(S_1, S_2, A, \delta, \mathsf{col})$ with $S_1 = S^1_1 \uplus S^2_1 \uplus \{t\} \setminus \{s_1, s_2\}$, $S_2 = S^1_2 \uplus S^2_2$; we define $A(t) = A^1(s_1) \uplus A^2(s_2)$ and for $i \in \{1, 2\}$, $\delta(t, a) = \delta^i(t, a)$ and $\mathsf{col}(t, a) = \mathsf{col}^i(t, a)$ if $a \in A(s_i)$; all the other available actions, transitions and colors are kept the same as in the original arenas (with transitions going to s_1 or s_2 being directed to t). A symmetrical definition can be written if s_1 and s_2 are both controlled by $\mathcal{P}_2$.

We can now present the two properties of preference relations at the core of our characterization. These properties are called X-Y-$\mathcal{M}$-*monotony* and X-Y-$\mathcal{M}$-*selectivity*; they depend on a type of strategies X, a type of arenas Y, and a memory skeleton $\mathcal{M}$. The first appearance of the monotony (resp. selectivity) notion was in [33], which dealt with deterministic arenas and memoryless strategies; their monotony (resp. selectivity) is equivalent to our P-D-$\mathcal{M}_{\mathsf{triv}}$-monotony (resp. P-D-$\mathcal{M}_{\mathsf{triv}}$-selectivity). In [7], these definitions were generalized to deal with the sufficiency of strategies based on $\mathcal{M}$ in deterministic arenas; their notion of $\mathcal{M}$-monotony (resp. $\mathcal{M}$-selectivity) is equivalent to our P-D-$\mathcal{M}$-monotony (resp. P-D-$\mathcal{M}$-selectivity).

▶ **Definition 9** (Monotony). *We say that* $\sqsubseteq$ *is* X-Y-$\mathcal{M}$-monotone *if for all* $m \in M$, *for all* $(\mathcal{A}_1, s_1), (\mathcal{A}_2, s_2) \in \mathfrak{A}^{\mathsf{Y}}_{\mathcal{P}_1}$, *there exists* $i \in \{1, 2\}$ *s.t. for all* $w \in L_{m_{\mathsf{init}}, m}$, $w[\mathcal{A}_{3-i}]^{\mathsf{X}}_{s_{3-i}} \sqsubseteq w[\mathcal{A}_i]^{\mathsf{X}}_{s_i}$.

The crucial part of the definition is the order of the last two quantifiers: of course, given a $w \in L_{m_{\mathsf{init}}, m}$, as $\sqsubseteq$ is total, it will always be the case that $w[\mathcal{A}_1]^{\mathsf{X}}_{s_1} \sqsubseteq w[\mathcal{A}_2]^{\mathsf{X}}_{s_2}$ or that $w[\mathcal{A}_2]^{\mathsf{X}}_{s_2} \sqsubseteq w[\mathcal{A}_1]^{\mathsf{X}}_{s_1}$. However, we ask for something stronger: it must be the case that the set of distributions $w[\mathcal{A}_i]^{\mathsf{X}}_{s_i}$ is preferred to $w[\mathcal{A}_{3-i}]^{\mathsf{X}}_{s_{3-i}}$ for *any* word $w \in L_{m_{\mathsf{init}}, m}$.

The original monotony definition [33] states that when presented with a choice *once* among two possible continuations, if a continuation is better than the other one after some prefix, then this continuation is also at least as good after all prefixes. This property is not sufficient for the sufficiency of pure memoryless strategies as it does not guarantee that if the same choice presents itself multiple times in the game, the same continuation should always be chosen, as alternating between both continuations might still be beneficial in the long run – this is dealt with by selectivity. If memory $\mathcal{M}$ is necessary to play optimally, then it makes sense that there are different optimal choices depending on the current memory state and that we should only compare prefixes that reach the same memory state. The point of taking into account a memory skeleton $\mathcal{M}$ in our definition of X-Y-$\mathcal{M}$-monotony is to only compare prefixes that are read up to the same memory state from m_{init}.

▶ **Definition 10** (Selectivity). *We say that* $\sqsubseteq$ *is* X-Y-$\mathcal{M}$-selective *if for all* $m \in M$, *for all* $(\mathcal{A}_1, s_1), (\mathcal{A}_2, s_2) \in \mathfrak{A}^{\mathsf{Y}}_{\mathcal{P}_1}$ *such that for* $i \in \{1, 2\}$, $\widehat{\mathsf{col}}(\mathsf{Hists}(\mathcal{A}_i, s_i, s_i)) \subseteq L_{m,m}$, *for all* $w \in L_{m_{\mathsf{init}}, m}$, $w[(\mathcal{A}_1, s_1) \sqcup (\mathcal{A}_2, s_2)]^{\mathsf{X}}_t \sqsubseteq w[\mathcal{A}_1]^{\mathsf{X}}_{s_1} \cup w[\mathcal{A}_2]^{\mathsf{X}}_{s_2}$ *(where* t *comes from the merge of* s_1 *and* s_2*).*

Our formulation of the selectivity concept differs from the original definition [33] and its AIFM counterpart [7] in order to take into account the particularities of the stochastic context, even if it can be proven that they are equivalent in the pure deterministic case. The idea is still the same: the original selectivity definition states that when presented with a choice among multiple possible continuations after some prefix, if a continuation is better than the others, then as the game goes on, if the same choice presents itself again, it is sufficient to always pick the same continuation to play optimally; there is no need to alternate between continuations. This property is not sufficient for the sufficiency of pure memoryless strategies as it does not guarantee that for all prefixes, the same initial choice is always the one we should commit to – this is dealt with by monotony. The point of memory skeleton $\mathcal{M}$ in our definition is to guarantee that every time the choice is presented, we are currently in the same memory state.

An interesting property is that both notions are stable by product with a memory skeleton: if $\sqsubseteq$ is X-Y-$\mathcal{M}$-monotone (resp. X-Y-$\mathcal{M}$-selective), then for all memory skeletons $\mathcal{M}'$, $\sqsubseteq$ is also X-Y-$(\mathcal{M} \otimes \mathcal{M}')$-monotone (resp. X-Y-$(\mathcal{M} \otimes \mathcal{M}')$-selective). The reason is that in each definition, we quantify universally over the class of all prefixes w that reach the same memory state m; if we consider classes that are subsets of the original classes, then the definition still holds. This property matches the idea that playing with more memory is never detrimental.

Combined together, it is intuitively reasonable that X-Y-$\mathcal{M}$-monotony and X-Y-$\mathcal{M}$-selectivity are equivalent to the sufficiency of pure strategies based on $\mathcal{M}$ to play X-optimally in $\mathfrak{A}^{\mathsf{Y}}_{\mathcal{P}_1}$: monotony tells us that when a single choice has to be made given a state of the arena and a memory state, the best choice is always the same no matter what prefix has been seen, and selectivity tells us that once a good choice has been made, we can commit to it in the future of the game. We formalize this idea in Theorem 13. First, we add an extra restriction on preference relations which is useful when stochasticity is involved.

▶ **Definition 11** (Mixing is useless). *We say that* mixing is useless for $\sqsubseteq$ *if for all sets I at most countable, for all positive reals $(\lambda_i)_{i\in I}$ such that $\sum_{i\in I} \lambda_i = 1$, for all families $(\mu_i)_{i\in I}$, $(\mu'_i)_{i\in I}$ of distributions in* $\mathsf{Dist}(C^\omega, \mathcal{F})$*, if for all $i \in I$, $\mu_i \sqsubseteq \mu'_i$, then $\sum_{i\in I} \lambda_i\mu_i \sqsubseteq \sum_{i\in I} \lambda_i\mu'_i$.*

That is, if we can write a distribution as a convex combination of distributions, then it is never detrimental to improve a distribution appearing in the convex combination.

▶ Remark 12. All preference relations encoded as Borel real payoff functions (as defined in Example 1) satisfy this property (it is easy to show the property for indicator functions, and we can then extend this fact to all Borel functions thanks to properties of the Lebesgue integral). The third preference relation from Example 1 (reaching $c \in C$ with probability precisely $\frac{1}{2}$) does not satisfy this property: if $\mu_1(\Diamond c) = 0$, $\mu'_1(\Diamond c) = \frac{1}{2}$, and $\mu_2(\Diamond c) = 1$, we have $\mu_1 \sqsubset \mu'_1$ and $\mu_2 \sqsubseteq \mu_2$, but $\frac{1}{2}\mu'_1 + \frac{1}{2}\mu_2 \sqsubset \frac{1}{2}\mu_1 + \frac{1}{2}\mu_2$. In deterministic games with pure strategies, only Dirac distributions on infinite words occur as distributions induced by an arena and a strategy, so the requirement that mixing is useless is not needed. ⌟

▶ **Theorem 13.** *Assume that no stochasticity is involved (that is, $\mathsf{X} \in \{\mathsf{P}, \mathsf{PFM}\}$ and $\mathsf{Y} = \mathsf{D}$), or that mixing is useless for $\sqsubseteq$. Then pure strategies based on $\mathcal{M}$ suffice to play X-optimally in all initialized one-player arenas in $\mathfrak{A}^{\mathsf{Y}}_{\mathcal{P}_1}$ for $\mathcal{P}_1$ if and only if $\sqsubseteq$ is X-Y-$\mathcal{M}$-monotone and X-Y-$\mathcal{M}$-selective.*

Proof sketch. We sketch both directions of the proof (available in [9]). The proof of the necessary condition of Theorem 13 is the easiest direction. The main idea is to build the right arenas (using the arenas occurring in the definitions of monotony and selectivity) so that we can use the hypothesis about the existence of pure X-optimal strategies based on $\mathcal{M}$ to immediately deduce X-Y-$\mathcal{M}$-monotony and X-Y-$\mathcal{M}$-selectivity. It is not necessary that mixing is useless for $\sqsubseteq$ for this direction of the equivalence.

For the sufficient condition, we first reduce as usual the statement to the existence of pure memoryless strategies in covered initialized arenas, using Lemma 6. We proceed with an edge-induction in these arenas (as for Theorem 8). The base case is trivial (as in an arena in which all states have a single available action, there is a single strategy which is pure and memoryless). For the induction step, we take an initialized arena $(\mathcal{A}', S_{\mathsf{init}}) \in \mathfrak{A}^{\mathsf{Y}}_{\mathcal{P}_1}$ covered by $\mathcal{M}$, and we pick a state t with (at least) two available actions. A memory state $\phi(t)$ is associated to t thanks to coverability. We consider arenas $(\mathcal{A}'_a, S_{\mathsf{init}})$ obtained from $(\mathcal{A}', S_{\mathsf{init}})$ by leaving a single action a available in t, to which we can apply the induction hypothesis and obtain a pure memoryless X-optimal strategy σ_1^a. It is left to prove that one of these strategies is also X-optimal in $(\mathcal{A}', S_{\mathsf{init}})$; this is where X-Y-$\mathcal{M}$-monotony and X-Y-$\mathcal{M}$-selectivity come into play.

The property of X-Y-$\mathcal{M}$-monotony tells us that one of these subarenas $(\mathcal{A}'_{a^*}, S_{\mathsf{init}})$ is preferred to the others w.r.t. $\sqsubseteq$ after reading any word in $L_{m_{\mathsf{init}},\phi(t)}$. We now want to use X-Y-$\mathcal{M}$-selectivity to conclude that there is no reason to use actions different from a^* when coming back to t, and that $\sigma_1^{a^*}$ is therefore also X-optimal in $(\mathcal{A}', S_{\mathsf{init}})$. To do so, we take any strategy $\sigma_1 \in \Sigma_1^{\mathsf{X}}(\mathcal{A}', s)$ for $s \in S_{\mathsf{init}}$ and we condition distribution $\mathsf{P}^{\sigma_1}_{\mathcal{A}',s}$ over all the ways it reaches (or not) t, which gives a convex combination of probability distributions. We want to state that once t is reached, no matter how, switching to strategy $\sigma_1^{a^*}$ is always beneficial. For this, we would like to use X-subgame-perfection of $\sigma_1^{a^*}$ rather than simply X-optimality: this is why in the actual proof, our induction hypothesis is about X-SP strategies and not X-optimal strategies. Luckily, Theorem 7 indicates that requiring subgame perfection is not really stronger than what we want to prove. We then use that mixing is useless for $\sqsubseteq$ (Definition 11) to replace all the parts that go through t in the convex combination by a better distribution induced by $\sigma_1^{a^*}$ from t. ◀

The literature provides some *sufficient* conditions for preference relations to admit pure memoryless optimal strategies in one-player stochastic games (for instance, in [31]). Here, we obtain a full characterization when mixing is useless for $\sqsubseteq$ (in particular, this is a full characterization for Borel real payoff functions), which can deal not only with memoryless strategies, but also with the more general AIFM strategies. It therefore provides a more fundamental understanding of preference relations for which AIFM strategies suffice or do not suffice. In particular, there are examples in which the known sufficient conditions are not verified even though pure memoryless strategies suffice (one such example is provided in [10]), and that is for instance where our characterization can help.

It is interesting to relate the concepts of monotony and selectivity to other properties from the literature to simplify the use of our characterization. For instance, if a real payoff function $f\colon C^\omega \to \mathbb{R}$ is *prefix-independent*,[1] then it is also X-Y-$\mathcal{M}$-monotone for any X, Y, and $\mathcal{M}$; therefore, the sufficiency of pure AIFM strategies immediately reduces to analyzing selectivity.

6 Conclusion

We have studied stochastic games and gave an overview of desirable properties of preference relations that admit *pure arena-independent finite-memory optimal strategies*. Our analysis provides general tools to help study memory requirements in stochastic games, both with one player (Markov decision processes) and two players, and links both problems. It generalizes both work on deterministic games [33, 7] and work on stochastic games [34].

[1] A function $f\colon C^\omega \to \mathbb{R}$ is *prefix-independent* if for all $w \in C^*$, for all $w' \in C^\omega$, $f(ww') = f(w')$.

A natural question that remains unsolved is the link between memory requirements of a preference relation in deterministic and in stochastic games; our results can be called independently to study both problems, but do not describe a bridge to go from one to the other yet. Also, our results can only be used to show the optimality of *pure* strategies with some fixed memory, but in some cases, using *randomized* strategies allows for lesser memory requirements [16, 41]. Investigating whether extensions to our results dealing with randomized strategies hold would therefore be valuable.

References

1 Benjamin Aminof and Sasha Rubin. First-cycle games. *Inf. Comput.*, 254:195–216, 2017. doi:10.1016/j.ic.2016.10.008.

2 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

3 Raphaël Berthon, Mickael Randour, and Jean-François Raskin. Threshold constraints with guarantees for parity objectives in Markov decision processes. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 121:1–121:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.121.

4 Alessandro Bianco, Marco Faella, Fabio Mogavero, and Aniello Murano. Exploring the boundary of half-positionality. *Ann. Math. Artif. Intell.*, 62(1-2):55–77, 2011. doi:10.1007/s10472-011-9250-1.

5 Patricia Bouyer, Ulrich Fahrenberg, Kim Guldstrand Larsen, Nicolas Markey, and Jirí Srba. Infinite runs in weighted timed automata with energy constraints. In Franck Cassez and Claude Jard, editors, *Formal Modeling and Analysis of Timed Systems, 6th International Conference, FORMATS 2008, Saint Malo, France, September 15-17, 2008. Proceedings*, volume 5215 of *Lecture Notes in Computer Science*, pages 33–47. Springer, 2008. doi:10.1007/978-3-540-85778-5_4.

6 Patricia Bouyer, Piotr Hofman, Nicolas Markey, Mickael Randour, and Martin Zimmermann. Bounding average-energy games. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 179–195, 2017. doi:10.1007/978-3-662-54458-7_11.

7 Patricia Bouyer, Stéphane Le Roux, Youssouf Oualhadj, Mickael Randour, and Pierre Vandenhove. Games where you can play optimally with arena-independent finite memory. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 24:1–24:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.24.

8 Patricia Bouyer, Nicolas Markey, Mickael Randour, Kim G. Larsen, and Simon Laursen. Average-energy games. *Acta Informatica*, 55(2):91–127, 2018. doi:10.1007/s00236-016-0274-1.

9 Patricia Bouyer, Youssouf Oualhadj, Mickael Randour, and Pierre Vandenhove. Arena-independent finite-memory determinacy in stochastic games. *CoRR*, abs/2102.10104, 2021. arXiv:2102.10104.

10 Tomás Brázdil, Václav Brozek, and Kousha Etessami. One-counter stochastic games. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 108–119. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.108.

11 Thomas Brihaye, Florent Delgrange, Youssouf Oualhadj, and Mickael Randour. Life is random, time is not: Markov decision processes with window objectives. *Log. Methods Comput. Sci.*, 16(4), 2020. URL: `https://lmcs.episciences.org/6975`.

12 Véronique Bruyère, Emmanuel Filiot, Mickael Randour, and Jean-François Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. *Inf. Comput.*, 254:259–295, 2017. `doi:10.1016/j.ic.2016.10.011`.

13 Véronique Bruyère, Quentin Hautem, and Mickael Randour. Window parity games: an alternative approach toward parity games with time bounds. In Domenico Cantone and Giorgio Delzanno, editors, *Proceedings of the Seventh International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2016, Catania, Italy, 14-16 September 2016*, volume 226 of *EPTCS*, pages 135–148, 2016. `doi:10.4204/EPTCS.226.10`.

14 Véronique Bruyère, Quentin Hautem, Mickael Randour, and Jean-François Raskin. Energy mean-payoff games. In Wan Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 21:1–21:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.CONCUR.2019.21`.

15 Krishnendu Chatterjee. The complexity of stochastic Müller games. *Inf. Comput.*, 211:29–48, 2012. `doi:10.1016/j.ic.2011.11.004`.

16 Krishnendu Chatterjee, Luca de Alfaro, and Thomas A. Henzinger. Trading memory for randomness. In *1st International Conference on Quantitative Evaluation of Systems (QEST 2004), 27-30 September 2004, Enschede, The Netherlands*, pages 206–217. IEEE Computer Society, 2004. `doi:10.1109/QEST.2004.1348035`.

17 Krishnendu Chatterjee and Laurent Doyen. Energy parity games. *Theor. Comput. Sci.*, 458:49–60, 2012. `doi:10.1016/j.tcs.2012.07.038`.

18 Krishnendu Chatterjee and Laurent Doyen. Perfect-information stochastic games with generalized mean-payoff objectives. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 247–256. ACM, 2016. `doi:10.1145/2933575.2934513`.

19 Krishnendu Chatterjee, Laurent Doyen, Hugo Gimbert, and Thomas A. Henzinger. Randomness for free. In Petr Hlinený and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2010. `doi:10.1007/978-3-642-15155-2_23`.

20 Krishnendu Chatterjee and Thomas A. Henzinger. A survey of stochastic ω-regular games. *J. Comput. Syst. Sci.*, 78(2):394–413, 2012. `doi:10.1016/j.jcss.2011.05.002`.

21 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Generalized parity games. In Helmut Seidl, editor, *Foundations of Software Science and Computational Structures, 10th International Conference, FOSSACS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 24-April 1, 2007, Proceedings*, volume 4423 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2007. `doi:10.1007/978-3-540-71389-0_12`.

22 Krishnendu Chatterjee, Marcin Jurdzinski, and Thomas A. Henzinger. Quantitative stochastic parity games. In J. Ian Munro, editor, *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2004, New Orleans, Louisiana, USA, January 11-14, 2004*, pages 121–130. SIAM, 2004. URL: `http://dl.acm.org/citation.cfm?id=982792.982808`.

23 Krishnendu Chatterjee, Joost-Pieter Katoen, Maximilian Weininger, and Tobias Winkler. Stochastic games with lexicographic reachability-safety objectives. In Shuvendu K. Lahiri and Chao Wang, editors, *Computer Aided Verification - 32nd International Conference, CAV 2020, Los Angeles, CA, USA, July 21-24, 2020, Proceedings, Part II*, volume 12225 of *Lecture Notes in Computer Science*, pages 398–420. Springer, 2020. `doi:10.1007/978-3-030-53291-8_21`.

24 Krishnendu Chatterjee, Zuzana Kretínská, and Jan Kretínský. Unifying two views on multiple mean-payoff objectives in Markov decision processes. *Log. Methods Comput. Sci.*, 13(2), 2017. doi:10.23638/LMCS-13(2:15)2017.

25 Krishnendu Chatterjee and Nir Piterman. Combinations of qualitative winning for stochastic parity games. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019, August 27-30, 2019, Amsterdam, the Netherlands*, volume 140 of *LIPIcs*, pages 6:1–6:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.6.

26 Krishnendu Chatterjee, Mickael Randour, and Jean-François Raskin. Strategy synthesis for multi-dimensional quantitative objectives. *Acta Inf.*, 51(3-4):129–163, 2014. doi:10.1007/s00236-013-0182-6.

27 Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, Aistis Simaitis, Ashutosh Trivedi, and Michael Ummels. Playing stochastic games precisely. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 348–363. Springer, 2012. doi:10.1007/978-3-642-32940-1_25.

28 Taolue Chen, Vojtech Forejt, Marta Z. Kwiatkowska, Aistis Simaitis, and Clemens Wiltsche. On stochastic games with multiple objectives. In Krishnendu Chatterjee and Jirí Sgall, editors, *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, volume 8087 of *Lecture Notes in Computer Science*, pages 266–277. Springer, 2013. doi:10.1007/978-3-642-40313-2_25.

29 Anne Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992. doi:10.1016/0890-5401(92)90048-K.

30 Florent Delgrange, Joost-Pieter Katoen, Tim Quatmann, and Mickael Randour. Simple strategies in multi-objective MDPs. In Armin Biere and David Parker, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I*, volume 12078 of *Lecture Notes in Computer Science*, pages 346–364. Springer, 2020. doi:10.1007/978-3-030-45190-5_19.

31 Hugo Gimbert. Pure stationary optimal strategies in Markov decision processes. In Wolfgang Thomas and Pascal Weil, editors, *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, volume 4393 of *Lecture Notes in Computer Science*, pages 200–211. Springer, 2007. doi:10.1007/978-3-540-70918-3_18.

32 Hugo Gimbert and Edon Kelmendi. Submixing and shift-invariant stochastic games. *CoRR*, abs/1401.6575, 2014. arXiv:1401.6575.

33 Hugo Gimbert and Wieslaw Zielonka. Games where you can play optimally without any memory. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, volume 3653 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2005. doi:10.1007/11539452_33.

34 Hugo Gimbert and Wieslaw Zielonka. Pure and Stationary Optimal Strategies in Perfect-Information Stochastic Games with Global Preferences. Unpublished, 2009. URL: https://hal.archives-ouvertes.fr/hal-00438359.

35 Eryk Kopczyński. Half-positional determinacy of infinite games. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 336–347. Springer, 2006. doi:10.1007/11787006_29.

36 Eryk Kopczyński. *Half-positional Determinacy of Infinite Games.* PhD thesis, Warsaw University, 2008.

37 Stéphane Le Roux and Arno Pauly. Extending finite-memory determinacy to multi-player games. *Inf. Comput.*, 261(Part):676–694, 2018. doi:10.1016/j.ic.2018.02.024.

38 Stéphane Le Roux, Arno Pauly, and Mickael Randour. Extending finite-memory determinacy by boolean combination of winning conditions. In Sumit Ganguly and Paritosh K. Pandya, editors, *38th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2018, December 11-13, 2018, Ahmedabad, India*, volume 122 of *LIPIcs*, pages 38:1–38:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.38.

39 Richard Mayr, Sven Schewe, Patrick Totzke, and Dominik Wojtczak. MDPs with energy-parity objectives. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005131.

40 Richard Mayr, Sven Schewe, Patrick Totzke, and Dominik Wojtczak. Simple stochastic games with almost-sure energy-parity objectives are in NP and coNP. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12650 of *Lecture Notes in Computer Science*, pages 427–447. Springer, 2021. doi:10.1007/978-3-030-71995-1_22.

41 Benjamin Monmege, Julie Parreaux, and Pierre-Alain Reynier. Reaching your goal optimally by playing at random with no memory. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 26:1–26:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.CONCUR.2020.26.

42 Andrzej W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In Andrzej Skowron, editor, *Computation Theory - Fifth Symposium, Zaborów, Poland, December 3-8, 1984, Proceedings*, volume 208 of *Lecture Notes in Computer Science*, pages 157–168. Springer, 1984. doi:10.1007/3-540-16066-3_15.

43 Martin J. Osborne. *An introduction to game theory.* Oxford University Press, 2004.

44 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley Series in Probability and Statistics. Wiley, 1994. doi:10.1002/9780470316887.

45 Mickael Randour. Automated synthesis of reliable and efficient systems through game theory: A case study. In *Proc. of ECCS 2012*, Springer Proceedings in Complexity XVII, pages 731–738. Springer, 2013. doi:10.1007/978-3-319-00395-5_90.

46 Mickael Randour, Jean-François Raskin, and Ocan Sankur. Variations on the stochastic shortest path problem. In Deepak D'Souza, Akash Lal, and Kim Guldstrand Larsen, editors, *Verification, Model Checking, and Abstract Interpretation - 16th International Conference, VMCAI 2015, Mumbai, India, January 12-14, 2015. Proceedings*, volume 8931 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2015. doi:10.1007/978-3-662-46081-8_1.

47 Mickael Randour, Jean-François Raskin, and Ocan Sankur. Percentile queries in multi-dimensional Markov decision processes. *Formal Methods Syst. Des.*, 50(2-3):207–248, 2017. doi:10.1007/s10703-016-0262-7.

48 L. S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953. doi:10.1073/pnas.39.10.1095.

49 Wolfgang Thomas. Church's problem and a tour through automata theory. In Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, editors, *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *Lecture Notes in Computer Science*, pages 635–655. Springer, 2008. doi:10.1007/978-3-540-78127-1_35.

50 Yaron Velner, Krishnendu Chatterjee, Laurent Doyen, Thomas A. Henzinger, Alexander Moshe Rabinovich, and Jean-François Raskin. The complexity of multi-mean-payoff and multi-energy games. *Inf. Comput.*, 241:177–196, 2015. doi:10.1016/j.ic.2015.03.001.

Stackelberg-Pareto Synthesis

Véronique Bruyère
Université de Mons (UMONS), Mons, Belgium

Jean-François Raskin
Université libre de Bruxelles (ULB), Brussels, Belgium

Clément Tamines
Université de Mons (UMONS), Mons, Belgium

Abstract

In this paper, we study the framework of two-player Stackelberg games played on graphs in which Player 0 announces a strategy and Player 1 responds rationally with a strategy that is an optimal response. While it is usually assumed that Player 1 has a single objective, we consider here the new setting where he has several. In this context, after responding with his strategy, Player 1 gets a payoff in the form of a vector of Booleans corresponding to his satisfied objectives. Rationality of Player 1 is encoded by the fact that his response must produce a Pareto-optimal payoff given the strategy of Player 0. We study the Stackelberg-Pareto Synthesis problem which asks whether Player 0 can announce a strategy which satisfies his objective, whatever the rational response of Player 1. For games in which objectives are either all parity or all reachability objectives, we show that this problem is fixed-parameter tractable and NEXPTIME-complete. This problem is already NP-complete in the simple case of reachability objectives and graphs that are trees.

2012 ACM Subject Classification Software and its engineering → Formal methods; Theory of computation → Logic and verification; Theory of computation → Solution concepts in game theory

Keywords and phrases Stackelberg non-zero sum games played on graphs, synthesis, parity objectives

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.27

Related Version *Full Version*: https://arxiv.org/abs/2102.08925 [9]

Funding This work is partially supported by the PDR project Subgame perfection in graph games (F.R.S.-FNRS), the ARC project Non-Zero Sum Game Graphs: Applications to Reactive Synthesis and Beyond (Fédération Wallonie-Bruxelles), the EOS project Verifying Learning Artificial Intelligence Systems (F.R.S.-FNRS and FWO), and the COST Action 16228 GAMENET (European Cooperation in Science and Technology).

1 Introduction

Two-player zero-sum infinite-duration games played on graphs are a mathematical model used to formalize several important problems in computer science, such as *reactive system synthesis*. In this context, see e.g. [26], the graph represents the possible interactions between the system and the environment in which it operates. One player models the system to synthesize, and the other player models the (uncontrollable) environment. In this classical setting, the objectives of the two players are opposite, that is, the environment is *adversarial*. Modelling the environment as fully adversarial is usually a *bold abstraction* of reality as it can be composed of one or several components, each of them having their own objective.

In this paper, we consider the framework of *Stackelberg games* [31], a richer non-zero-sum setting, in which Player 0 (the system) called *leader* announces his strategy and then Player 1 (the environment) called *follower* plays rationally by using a strategy that is an optimal response to the leader's strategy. This framework captures the fact that in practical applications, a strategy for interacting with the environment is committed before the interaction actually happens. The goal of the leader is to announce a strategy that

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 27; pp. 27:1–27:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

guarantees him a payoff at least equal to some given threshold. In the specific case of Boolean objectives, the leader wants to see his objective being satisfied. The concept of leader and follower is also present in the framework of *rational synthesis* [17, 24] with the difference that this framework considers several followers, each of them with their own Boolean objective. In that case, rationality of the followers is modeled by assuming that the environment settles to an equilibrium (e.g. a Nash equilibrium) where each component (composing the environment) is considered to be an *independent selfish individual*, excluding cooperation scenarios between components or the possibility of coordinated rational multiple deviations. Our work proposes a novel and natural *alternative* in which the single follower, modeling the environment, has several objectives that he wants to satisfy. After responding to the leader with his own strategy, Player 1 receives a vector of Booleans which is his payoff in the corresponding outcome. Rationality of Player 1 is encoded by the fact that he only responds in such a way to receive *Pareto-optimal payoffs*, given the strategy announced by the leader. This setting encompasses scenarios where, for instance, several components can collaborate and agree on trade-offs. The goal of the leader is therefore to announce a strategy that guarantees him to satisfy his own objective, whatever the response of the follower which ensures him a Pareto-optimal payoff. The problem of deciding whether the leader has such a strategy is called the *Stackelberg-Pareto Synthesis problem* (SPS problem).

Contributions. In addition to the definition of the new setting, our main contributions are the following ones. We consider the general class of ω-regular objectives modelled by *parity* conditions and also consider the case of *reachability* objectives for their simplicity[1]. We provide a thorough analysis of the complexity of solving the SPS problem for both objectives. Our results are interesting and singular both from a theoretical and practical point of view.

First, we show that the SPS problem is *fixed-parameter tractable* (FPT) for reachability objectives when the number of objectives of the follower is a parameter and for parity objectives when, in addition, the maximal priority used in each priority function is also a parameter of the complexity analysis (Theorem 3). These are important results as it is expected that, in practice, the *number* of objectives of the environment is limited to a few. To obtain these results, we develop a reduction from our non-zero-sum games to a zero-sum game in which the protagonist, called *Prover*, tries to show the existence of a solution to the problem, while the antagonist, called *Challenger*, tries to disprove it. This zero-sum game is defined in a *generic way*, independently of the actual objectives used in the initial game, and can then be easily adapted according to the case of reachability or parity objectives.

Second, we prove that the SPS problem is NEXPTIME-complete for both reachability and parity objectives (Theorem 6 and Theorem 9), and that it is already NP-complete in the simple setting of reachability objectives and graphs that are trees (Theorem 7). To the best of our knowledge, this is the first NEXPTIME-completeness result for a natural class of games played on graphs. To obtain the hardness for NEXPTIME, we present a natural *succinct version* of the set cover problem that is complete for this class (Theorem 11), a result of potential independent interest. We then show how to reduce this problem to the SPS problem. To obtain the NEXPTIME-membership of the SPS problem, we have shown that exponential-size solutions exist for positive instances of the SPS problem and this allows us to design a nondeterministic exponential-time algorithm. Unfortunately, it was not possible to use the FPT algorithm mentioned above to show this membership due to its too high time complexity; conversely, our NEXPTIME algorithm is not FPT.

[1] Indeed, in the classical context of two-player zero-sum games, solving reachability games is in P whereas solving parity games is only known to be in NP $\cap$ co-NP, see e.g. [18].

Related Work. Rational synthesis is introduced in [17] for ω-regular objectives in a setting where the followers are cooperative with the leader, and later in [24] where they are adversarial. Precise complexity results for various ω-regular objectives are established in [13] for both settings. Those complexities differ from the ones of the problem studied in this paper. Indeed, for reachability objectives, adversarial rational synthesis is PSPACE-complete, while for parity objectives, its precise complexity is not settled (the problem is PSPACE-hard and in NEXPTIME). Extension to non-Boolean payoffs, like mean-payoff or discounted sum, is studied in [19, 20] in the cooperative setting and in [1, 16] in the adversarial setting.

When several players (like the followers) play with the aim to satisfy their objectives, several solution concepts exist such as Nash equilibrium [25], subgame perfect equilibrium [27], secure equilibria [11, 12], or admissibility [2, 4]. The constrained existence problem, close to the cooperative rational synthesis problem, is to decide whether there exists a solution concept such that the payoff obtained by each player is larger than some threshold. Let us mention [13, 29, 30] for results on the constrained existence for Nash equilibria and [5, 6, 28] for such results for subgame perfect equilibria. Rational verification is studied in [21, 22]. This problem (which is not a synthesis problem) is to decide whether a given LTL formula is satisfied by the outcome of all Nash equilibria (resp. some Nash equilibrium). The interested reader can find more pointers to works on non-zero-sum games for reactive synthesis in [3, 7].

Structure. The paper is structured as follows. In Section 2, we introduce the class of Stackelberg-Pareto games and the SPS problem. We show in Section 3 that the SPS problem is in FPT for reachability and parity objectives. The complexity class of this problem is studied in Section 4 where we prove that it is NEXPTIME-complete and NP-complete in case of reachability objectives and graphs that are trees. In Section 5, we provide a conclusion and discuss future work. Detailed proofs of our results can be found in the full version of this paper.

2 Preliminaries and Stackelberg-Pareto Synthesis Problem

This section introduces the class of two-player Stackelberg-Pareto games in which the first player has a single objective and the second has several. We present a decision problem on those games called the Stackelberg-Pareto Synthesis problem, which we study in this paper.

2.1 Preliminaries

Game Arena. A *game arena* is a tuple $G = (V, V_0, V_1, E, v_0)$ where (V, E) is a finite directed graph such that: *(i)* V is the set of vertices and (V_0, V_1) forms a partition of V where V_0 (resp. V_1) is the set of vertices controlled by Player 0 (resp. Player 1), *(ii)* $E \subseteq V \times V$ is the set of edges such that each vertex v has at least one successor v', i.e., $(v, v') \in E$, and *(iii)* $v_0 \in V$ is the initial vertex. We call a game arena a *tree arena* if it is a tree in which every leaf vertex has itself as its only successor. A *sub-arena* G' with a set $V' \subseteq V$ of vertices and initial vertex $v'_0 \in V'$ is a game arena defined from G as expected.

Plays. A *play* in a game arena G is an infinite sequence of vertices $\rho = v_0 v_1 \ldots \in V^\omega$ such that it starts with the initial vertex v_0 and $(v_j, v_{j+1}) \in E$ for all $j \in \mathbb{N}$. *Histories* in G are finite sequences $h = v_0 \ldots v_j \in V^+$ defined similarly. A history is *elementary* if it contains no cycles. We denote by Plays_G the set of plays in G. We write Hist_G (resp. $\mathsf{Hist}_{G,i}$) the set of histories (resp. histories ending with a vertex in V_i). We use the notations Plays, Hist, and Hist_i when G is clear from the context. We write $\mathsf{Occ}(\rho)$ the set of vertices occurring in ρ and $\mathsf{Inf}(\rho)$ the set of vertices occurring infinitely often in ρ.

Strategies. A *strategy* σ_i for Player i is a function $\sigma_i \colon \mathsf{Hist}_i \to V$ assigning to each history $hv \in \mathsf{Hist}_i$ a vertex $v' = \sigma_i(hv)$ such that $(v, v') \in E$. It is *memoryless* if $\sigma_i(hv) = \sigma_i(h'v)$ for all histories $hv, h'v$ ending with the same vertex $v \in V_i$. More generally, it is *finite-memory* if it can be encoded by a Moore machine $\mathcal{M}$ [18]. The *memory size* of σ_i is the number of memory states of $\mathcal{M}$. In particular, σ_i is memoryless when it has a memory size of one.

Given a strategy σ_i of Player i, a play $\rho = v_0 v_1 \dots$ is *consistent* with σ_i if $v_{j+1} = \sigma_i(v_0 \dots v_j)$ for all $j \in \mathbb{N}$ such that $v_j \in V_i$. Consistency is naturally extended to histories. We denote by $\mathsf{Plays}_{\sigma_i}$ (resp. Hist_{σ_i}) the set of plays (resp. histories) consistent with σ_i. A *strategy profile* is a tuple $\sigma = (\sigma_0, \sigma_1)$ of strategies, one for each player. We write $\mathsf{out}(\sigma)$ the unique play consistent with both strategies and we call it the *outcome* of σ.

Objectives. An *objective* for Player i is a set of plays $\Omega \subseteq \mathsf{Plays}$. A play ρ *satisfies* the objective Ω if $\rho \in \Omega$. In this paper, we focus on the two following ω-regular objectives. Let $T \subseteq V$ be a subset of vertices called a *target set*, the *reachability* objective $\mathsf{Reach}(T) = \{\rho \in \mathsf{Plays} \mid \mathsf{Occ}(\rho) \cap T \neq \emptyset\}$ asks to visit at least one vertex of T. Let $c : V \to \mathbb{N}$ be a function called a *priority function* which assigns an integer to each vertex in the arena, the *parity* objective $\mathsf{Parity}(c) = \{\rho \in \mathsf{Plays} \mid \min_{v \in \mathsf{Inf}(\rho)}(c(v)) \text{ is even}\}$ asks that the minimum priority visited infinitely often be even.

2.2 Stackelberg-Pareto Synthesis Problem

Stackelberg-Pareto Games. A *Stackelberg-Pareto game* (SP game) $\mathcal{G} = (G, \Omega_0, \Omega_1, \dots, \Omega_t)$ is composed of a game arena G, an objective Ω_0 for Player 0 and $t \geq 1$ objectives $\Omega_1, \dots, \Omega_t$ for Player 1. In this paper, we focus on SP games where the objectives are either all reachability or all parity objectives and call such games *reachability* (resp. *parity*) *SP games*.

Payoffs in SP Games. The *payoff* of a play $\rho \in \mathsf{Plays}$ corresponds to the vector of Booleans $\mathsf{pay}(\rho) \in \{0,1\}^t$ such that for all $i \in \{1, \dots, t\}$, $\mathsf{pay}_i(\rho) = 1$ if $\rho \in \Omega_i$, and $\mathsf{pay}_i(\rho) = 0$ otherwise. Note that we omit to include Player 0 when discussing the payoff of a play. Instead we say that a play ρ is *won* by Player 0 if $\rho \in \Omega_0$ and we write $\mathsf{won}(\rho) = 1$, otherwise it is *lost* by Player 0 and we write $\mathsf{won}(\rho) = 0$. We write $(\mathsf{won}(\rho), \mathsf{pay}(\rho))$ the *extended payoff* of ρ. Given a strategy profile σ, we write $\mathsf{won}(\sigma) = \mathsf{won}(\mathsf{out}(\sigma))$ and $\mathsf{pay}(\sigma) = \mathsf{pay}(\mathsf{out}(\sigma))$. For reachability SP games, since reachability objectives are prefix-dependant and given a history $h \in \mathsf{Hist}$, we also define $\mathsf{won}(h)$ and $\mathsf{pay}(h)$ as done for plays.

We introduce the following partial order on payoffs. Given two payoffs $p = (p_1, \dots, p_t)$ and $p' = (p'_1, \dots, p'_t)$ such that $p, p' \in \{0,1\}^t$, we say that p' is *larger* than p and write $p \leq p'$ if $p_i \leq p'_i$ for all $i \in \{1, \dots, t\}$. Moreover, when it also holds that $p_i < p'_i$ for some i, we say that p' is *strictly larger* than p and we write $p < p'$. A subset of payoffs $P \subseteq \{0,1\}^t$ is an *antichain* if it is composed of pairwise incomparable payoffs with respect to $\leq$.

Stackelberg-Pareto Synthesis Problem. Given a strategy σ_0 of Player 0, we consider the set of payoffs of plays consistent with σ_0 which are *Pareto-optimal*, i.e., maximal with respect to $\leq$. We write this set $P_{\sigma_0} = \max\{\mathsf{pay}(\rho) \mid \rho \in \mathsf{Plays}_{\sigma_0}\}$. Notice that it is an antichain. We say that those payoffs are σ_0-*fixed Pareto-optimal* and write $|P_{\sigma_0}|$ the number of such payoffs. A play $\rho \in \mathsf{Plays}_{\sigma_0}$ is called σ_0-fixed Pareto-optimal if its payoff $\mathsf{pay}(\rho)$ is in P_{σ_0}.

The problem studied in this paper asks whether there exists a strategy σ_0 for Player 0 such that every play in $\mathsf{Plays}_{\sigma_0}$ which is σ_0-fixed Pareto-optimal satisfies the objective of Player 0. This corresponds to the assumption that given a strategy of Player 0, Player 1 will play *rationally*, that is, with a strategy σ_1 such that $\mathsf{out}((\sigma_0, \sigma_1))$ is σ_0-fixed Pareto-optimal. It is therefore sound to ask that Player 0 wins against such rational strategies.

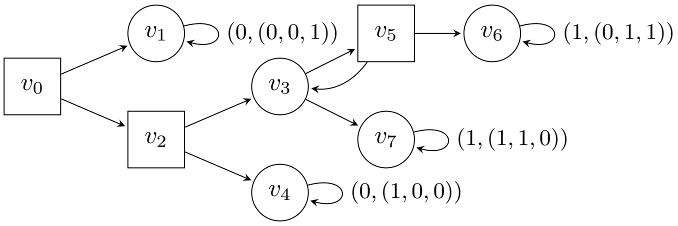


Figure 1 A reachability SP game.

▶ **Definition 1.** *Given an SP game, the* Stackelberg-Pareto Synthesis problem *(SPS problem) is to decide whether there exists a strategy σ_0 for Player 0 (called a* solution*) such that for each strategy profile $\sigma = (\sigma_0, \sigma_1)$ with $\mathsf{pay}(\sigma) \in P_{\sigma_0}$, it holds that $\mathsf{won}(\sigma) = 1$.*

Witnesses. Given a strategy σ_0 that is a solution to the SPS problem and any payoff $p \in P_{\sigma_0}$, for each play ρ consistent with σ_0 such that $\mathsf{pay}(\rho) = p$ it holds that $\mathsf{won}(\rho) = 1$. For each $p \in P_{\sigma_0}$, we arbitrarily select such a play which we call a *witness* (of p). We denote by Wit_{σ_0} the set of all witnesses, of which there are as many as payoffs in P_{σ_0}. In the sequel, it is useful to see this set as a tree composed of $|\mathsf{Wit}_{\sigma_0}|$ branches. Additionally for a given history $h \in \mathsf{Hist}$, we write $\mathsf{Wit}_{\sigma_0}(h)$ the set of witnesses for which h is a prefix, i.e., $\mathsf{Wit}_{\sigma_0}(h) = \{\rho \in \mathsf{Wit}_{\sigma_0} \mid h \text{ is prefix of } \rho\}$. Notice that $\mathsf{Wit}_{\sigma_0}(h) = \mathsf{Wit}_{\sigma_0}$ when $h = v_0$ and that $\mathsf{Wit}_{\sigma_0}(h)$ decreases as h increases, until it contains a single value or becomes empty.

▶ **Example 2.** Consider the reachability SP game with arena G depicted in Figure 1 in which Player 1 has $t = 3$ objectives. The vertices of Player 0 (resp. Player 1) are depicted as ellipses (resp. rectangles)[2]. Every objective in the game is a reachability objective defined as follows: $\Omega_0 = \mathsf{Reach}(\{v_6, v_7\})$, $\Omega_1 = \mathsf{Reach}(\{v_4, v_7\})$, $\Omega_2 = \mathsf{Reach}(\{v_3\})$, $\Omega_3 = \mathsf{Reach}(\{v_1, v_6\})$. The extended payoff of plays reaching vertices from which they can only loop is displayed in the arena next to those vertices, and the extended payoff of play $v_0v_2(v_3v_5)^\omega$ is $(0,(0,1,0))$.

Consider the memoryless strategy σ_0 of Player 0 such that he chooses to always move to v_5 from v_3. The set of payoffs of plays consistent with σ_0 is $\{(0,0,1),(0,1,0),(1,0,0),(0,1,1)\}$ and the set of those that are Pareto-optimal is $P_{\sigma_0} = \{(1,0,0),(0,1,1)\}$. Notice that play $\rho = v_0v_2(v_4)^\omega$ is consistent with σ_0, has payoff $(1,0,0)$ and is lost by Player 0. Strategy σ_0 is therefore not a solution to the SPS problem. In this game, there is only one other memoryless strategy for Player 0, where he chooses to always move to v_7 from v_3. One can verify that it is again not a solution to the SPS problem.

We can however define a finite-memory strategy σ_0' such that $\sigma_0'(v_0v_2v_3) = v_5$ and $\sigma_0'(v_0v_2v_3v_5v_3) = v_7$ and show that it is a solution to the problem. Indeed, the set of σ_0'-fixed Pareto-optimal payoffs is $P_{\sigma_0'} = \{(0,1,1),(1,1,0)\}$ and Player 0 wins every play consistent with σ_0' whose payoff is in this set. A set $\mathsf{Wit}_{\sigma_0'}$ of witnesses for these payoffs is $\{v_0v_2v_3v_5v_6^\omega, v_0v_2v_3v_5v_3v_7^\omega\}$ and is in this case the unique set of witnesses. This example shows that Player 0 sometimes needs memory in order to have a solution to the SPS problem.

3 Fixed-Parameter Complexity

In this section, we show that the SPS problem is in FPT for both cases of reachability and parity SP games. We refer the reader to [15] for the concept of fixed-parameter complexity.

[2] This convention is used throughout this paper.

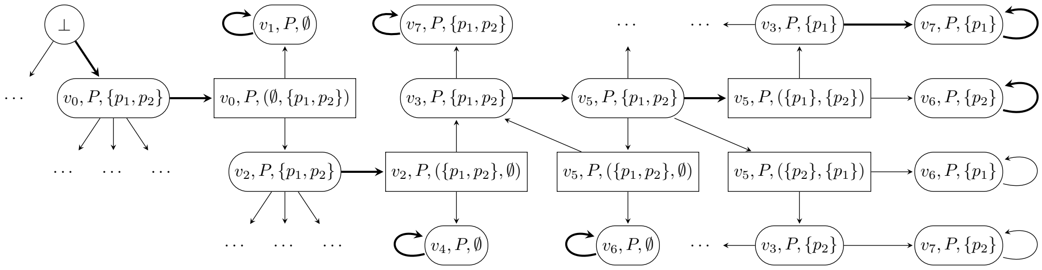


Figure 2 A part of the C-P game for Example 2 with $P = \{p_1, p_2\}, p_1 = (1, 1, 0)$, and $p_2 = (0, 1, 1)$.

▶ **Theorem 3.** *Solving the SPS problem is in* FPT *for reachability SP games for parameter t equal to the number of objectives of Player 1 and it is in* FPT *for parity SP games for parameters t and the maximal priority according to each parity objective of Player 1.*

3.1 Challenger-Prover Game

In order to prove Theorem 3, we provide a reduction to a specific two-player zero-sum game, called the *Challenger-Prover* game (C-P game). This game is a *zero-sum*[3] game played between *Challenger* (written $\mathcal{C}$) and *Prover* (written $\mathcal{P}$). We will show that Player 0 has a solution to the SPS problem in an SP game if and only if $\mathcal{P}$ has a winning strategy in the corresponding C-P game. In the latter game, $\mathcal{P}$ tries to show the existence of a strategy σ_0 that is solution to the SPS problem in the original game and $\mathcal{C}$ tries to disprove it. The C-P game is described independently of the objectives used in the SP game and its objective is described as such in a *generic way*. We later provide the proof of our FPT results by adapting it specifically for reachability and parity SP games.

Intuition on the C-P Game. Without loss of generality, the SP games we consider in this section are such that each vertex in their arena has at most *two successors*. It can be shown that any SP game $\mathcal{G}$ with n vertices can be transformed into an SP game $\bar{\mathcal{G}}$ with $\mathcal{O}(n^2)$ vertices such that every vertex has at most two successors and Player 0 has a solution to the SPS problem in $\mathcal{G}$ if and only if he has a solution to the SPS problem in $\bar{\mathcal{G}}$.

Let $\mathcal{G}$ be an SP game. The C-P game $\mathcal{G}'$ is a zero-sum game associated with $\mathcal{G}$ that intuitively works as follows. First, $\mathcal{P}$ selects a set P of payoffs which he announces as the set of Pareto-optimal payoffs P_{σ_0} for the solution σ_0 to the SPS problem in $\mathcal{G}$ he is trying to construct. Then, $\mathcal{P}$ tries to show that there exists a set of witnesses Wit_{σ_0} in $\mathcal{G}$ for the payoffs in P. After the selection of P in $\mathcal{G}'$, there is a one-to-one correspondence between plays in the arenas G and G' such that the vertices in G' are augmented with a set W which is a subset of P. Initially W is equal to P and after some history in G', W contains payoff p if the corresponding history in G is prefix of the witness with payoff p in the set Wit_{σ_0} that $\mathcal{P}$ is building. In addition, the objective $\Omega_{\mathcal{P}}$ of $\mathcal{P}$ is such that he has a winning strategy $\sigma_{\mathcal{P}}$ in $\mathcal{G}'$ if and only if the set P that he selected coincides with the set P_{σ_0} for the corresponding strategy σ_0 in $\mathcal{G}$ and the latter strategy is a solution to the SPS problem in $\mathcal{G}$. A part of the arena of the C-P game for Example 2 with a positional winning strategy for $\mathcal{P}$ highlighted in bold is illustrated in Figure 2.

[3] We assume that the reader is familiar with the concept of zero-sum games, see e.g. [18].

Arena of the C-P Game. The initial vertex $\bot$ belongs to $\mathcal{P}$. From this vertex, he selects a successor (v_0, P, W) such that $W = P$ and P is an antichain of payoffs which $\mathcal{P}$ announces as the set P_{σ_0} for the strategy σ_0 in G he is trying to construct. All vertices in plays starting with this vertex will have this same value for their P-component. Those vertices are either a triplet (v, P, W) that belongs to $\mathcal{P}$ or $(v, P, (W_l, W_r))$ that belongs to $\mathcal{C}$. Given a play ρ (resp. history h) in G', we denote by ρ_V (resp. h_V) the play (resp. history) in G obtained by removing $\bot$ and keeping the v-component of every vertex of $\mathcal{P}$ in ρ (resp. h), which we call its *projection*.

- After history hm such that $m = (v, P, W)$ with $v \in V_0$, $\mathcal{P}$ selects a successor v' such that $(v, v') \in E$ and vertex (v', P, W) is added to the play. This corresponds to Player 0 choosing a successor v' after history $h_V v$ in G.
- After history hm such that $m = (v, P, W)$ with $v \in V_1$, $\mathcal{P}$ selects a successor $(v, P, (W_l, W_r))$ with (W_l, W_r) a partition of W. This corresponds to $\mathcal{P}$ splitting the set W into two parts according to the two successors v_l and v_r of v. For the strategy σ_0 that $\mathcal{P}$ tries to construct and its set of witnesses Wit_{σ_0} he is building, he asserts that W_l (resp. W_r) is the set of payoffs of the witnesses in $\mathsf{Wit}_{\sigma_0}(h_V v_l)$ (resp. $\mathsf{Wit}_{\sigma_0}(h_V v_r)$).
- From a vertex $(v, P, (W_l, W_r))$, $\mathcal{C}$ can select a successor (v_l, P, W_l) or (v_r, P, W_r) which corresponds to the choice of Player 1.

Formally, the game arena of the C-P game is the tuple $G' = (V', V'_{\mathcal{P}}, V'_{\mathcal{C}}, E', \bot)$ with

- $V'_{\mathcal{P}} = \{\bot\} \cup \{(v, P, W) \mid v \in V, P \subseteq \{0,1\}^t$ is an antichain and $W \subseteq P\}$,
- $V'_{\mathcal{C}} = \{(v, P, (W_l, W_r)) \mid v \in V_1, P \subseteq \{0,1\}^t$ is an antichain and $W_l, W_r \subseteq P\}$,
- $(\bot, (v, P, W)) \in E'$ if $v = v_0$ and $P = W$,
- $((v, P, W), (v', P, W)) \in E'$ if $v \in V_0$ and $(v, v') \in E$,
- $((v, P, W), (v, P, (W_l, W_r))) \in E'$ if $v \in V_1$ and (W_l, W_r) is a partition of W,
- $((v, P, (W_l, W_r)), (v', P, W)) \in E'$ if $(v, v') \in E$ and $\{v' = v_l$ and $W = W_l\}$ or $\{v' = v_r$ and $W = W_r\}$.

In the definition of E', if v has a single successor v' in G, it is assumed to be v_l and W_r is always equal to $\emptyset$. Given the two successors v_i and v_j of v, v_i is the left successor if $i < j$.

Objective of $\mathcal{P}$ in the C-P Game. Let us now discuss the objective $\Omega_{\mathcal{P}}$ of $\mathcal{P}$. The W-component of the vertices controlled by $\mathcal{P}$ has a size that decreases along a play ρ in G'. We write $lim_W(\rho)$ the value of the W-component at the limit in ρ. Recall that with this W-component, $\mathcal{P}$ tries to construct a solution σ_0 to the SPS problem with associated sets P_{σ_0} and Wit_{σ_0}. Therefore, for him to win in the C-P game, $lim_W(\rho)$ must be a singleton or empty in every consistent play such that:

- $lim_W(\rho)$ must be a singleton $\{p\}$ with p the payoff of ρ_V in G, showing that $\rho_V \in \mathsf{Wit}_{\sigma_0}$ is a correct witness for p. In addition, it must hold that $\mathsf{won}(\rho_V) = 1$ as $p \in P$ and as $\mathcal{P}$ wants σ_0 to be a solution.
- $lim_W(\rho)$ must be the empty set such that either the payoff of ρ_V belongs to P_{σ_0} and $\mathsf{won}(\rho_V) = 1$, or the payoff of ρ_V is strictly smaller than some payoff in P_{σ_0}.

These conditions verify that the sets $P = P_{\sigma_0}$ and Wit_{σ_0} are correct and that σ_0 is indeed a solution to the SPS problem in G. They are generic as they do not depend on the actual objectives used in the SP game.

Let us give the formal definition of $\Omega_{\mathcal{P}}$. For an antichain P of payoffs, we write $\mathsf{Plays}^P_{G'}$ the set of plays in G' which start with $\bot(v_0, P, P)$ and we define the following set

$$\begin{aligned} B_P = \big\{\rho \in \mathsf{Plays}^P_{G'} \mid\ & (lim_W(\rho) = \{p\} \wedge \mathsf{pay}(\rho_V) = p \in P \wedge \mathsf{won}(\rho_V) = 1) \vee && (1)\\ & (lim_W(\rho) = \emptyset \quad \wedge \mathsf{pay}(\rho_V) \in P \qquad \wedge \mathsf{won}(\rho_V) = 1) \vee && (2)\\ & (lim_W(\rho) = \emptyset \quad \wedge \exists p \in P,\ \mathsf{pay}(\rho_V) < p)\big\}. && (3)\end{aligned}$$

Objective $\Omega_{\mathcal{P}}$ of $\mathcal{P}$ in $\mathcal{G}'$ is the union of B_P over all antichains P. As the C-P game is zero-sum, objective $\Omega_{\mathcal{C}}$ equals $\mathsf{Plays}_{G'} \setminus \Omega_{\mathcal{P}}$. The following theorem holds.

▶ **Theorem 4.** *Player 0 has a strategy σ_0 that is solution to the SPS problem in $\mathcal{G}$ if and only if $\mathcal{P}$ has a winning strategy $\sigma_{\mathcal{P}}$ from $\bot$ in the C-P game $\mathcal{G}'$.*

Proof. Let us first assume that Player 0 has a strategy σ_0 that is solution to the SPS problem in $\mathcal{G}$. Let P_{σ_0} be its set of σ_0-fixed Pareto-optimal payoffs and let Wit_{σ_0} be a set of witnesses. We construct the strategy $\sigma_{\mathcal{P}}$ from σ_0 such that

- $\sigma_{\mathcal{P}}(\bot) = (v_0, P, P)$ such that $P = P_{\sigma_0}$ (this vertex exists as P_{σ_0} is an antichain),
- $\sigma_{\mathcal{P}}(hm) = (v', P, W)$ if $m = (v, P, W)$ with $v \in V_0$ and $v' = \sigma_0(h_V v)$,
- $\sigma_{\mathcal{P}}(hm) = (v, P, (W_l, W_r))$ if $m = (v, P, W)$ with $v \in V_1$ and for $i \in \{l, r\}$, $W_i = \{\mathsf{pay}(\rho) \mid \rho \in \mathsf{Wit}_{\sigma_0}(h_V v_i)\}$.

It is clear that given a play ρ in G' consistent with $\sigma_{\mathcal{P}}$, the play ρ_V in G is consistent with σ_0. Let us show that $\sigma_{\mathcal{P}}$ is winning for $\mathcal{P}$ from $\bot$ in G'. Consider a play ρ in G' consistent with $\sigma_{\mathcal{P}}$. There are two possibilities. *(i)* ρ_V is a witness of Wit_{σ_0} and by construction $lim_W(\rho) = \{p\}$ with $p = \mathsf{pay}(\rho_V)$; thus $\mathsf{won}(\rho_V) = 1$ as σ_0 is a solution and ρ_V is a witness. *(ii)* ρ_V is not a witness and by construction $lim_W(\rho) = \emptyset$; as σ_0 is a solution, then $p = \mathsf{pay}(\rho_V)$ is bounded by some payoff of P_{σ_0} and in case of equality $\mathsf{won}(\rho_V) = 1$. Therefore ρ satisfies the objective B_P of $\Omega_{\mathcal{P}}$ since it satisfies condition (1) in case *(i)* and condition (2) or (3) in case *(ii)*.

Let us now assume that $\mathcal{P}$ has a winning strategy $\sigma_{\mathcal{P}}$ from $\bot$ in G'. Let P be the antichain of payoffs chosen from $\bot$ by this strategy. We construct the strategy σ_0 from $\sigma_{\mathcal{P}}$ such that $\sigma_0(h_V v) = v'$ given $\sigma_{\mathcal{P}}(hm) = (v', P, W)$ with $m = (v, P, W)$ and $v \in V_0$. Notice that this definition makes sense since there is a unique history hm ending with a vertex of $\mathcal{P}$ associated with $h_V v$ showing a one-to-one correspondence between those histories.

Let us show σ_0 is a solution to the SPS problem with P_{σ_0} being the set P. First notice that P is not empty. Indeed let ρ be a play consistent with $\sigma_{\mathcal{P}}$. As ρ belongs to $\Omega_{\mathcal{P}}$ and in particular to B_P, one can check that $P \neq \emptyset$ by inspecting conditions (1) to (3). Second notice that by definition of E', if $((v, P, W), (v, P, (W_l, W_r))) \in E'$ with $W \neq \emptyset$, then either W_l or W_r is not empty. Therefore given any payoff $p \in P$, there is a unique play ρ consistent with $\sigma_{\mathcal{P}}$ such that $lim_W(\rho) = \{p\}$. By construction of σ_0 and as $\sigma_{\mathcal{P}}$ is winning, the play ρ_V is consistent with σ_0, has payoff p, and is won by Player 0 (see (1)).

Let ρ_V be a play consistent with σ_0 and ρ be the corresponding play consistent with $\sigma_{\mathcal{P}}$. It remains to consider (2) and (3). These conditions indicate that ρ_V has a payoff equal to or strictly smaller than a payoff in P and that in case of equality $\mathsf{won}(\rho_V) = 1$. This shows that $P_{\sigma_0} = P$ and that σ_0 is a solution to the SPS problem. ◀

3.2 Proof of the FPT Results

We now sketch the proof of Theorem 3 which works by specializing the generic objective $\Omega_{\mathcal{P}}$ to handle reachability and parity SP games. We begin with reachability SP games. We extend the arena G' of the C-P game such that its vertices keep track of the objectives of $\mathcal{G}$ which are satisfied along a play. Given an extended payoff $(w, p) \in \{0,1\} \times \{0,1\}^t$ and a vertex $v \in V$, we define the *payoff update* $\mathsf{upd}(w, p, v) = (w', p')$ such that

$$\begin{aligned} w' = 1 &\iff w = 1 \text{ or } v \in T_0, \\ p'_i = 1 &\iff p_i = 1 \text{ or } v \in T_i, \quad \forall i \in \{1, \dots, t\}. \end{aligned}$$

We obtain the extended arena G^* as follows: *(i)* its set of vertices is $V' \times \{0,1\} \times \{0,1\}^t$, *(ii)* its initial vertex is $\bot^* = (\bot, 0, (0, \dots, 0))$, and *(iii)* $((m, w, p), (m', w', p'))$ with $m' = (v', P, W)$ or $m' = (v', P, (W_l, W_r))$ is an edge in G^* if $(m, m') \in E'$ and $(w', p') = \mathsf{upd}(w, p, v')$.

We define the zero-sum game $\mathcal{G}^* = (G^*, \Omega^*_{\mathcal{P}})$ in which the three abstract conditions (1-3) detailed previously are encoded into the following Büchi objective by using the (w, p)-component added to vertices. We define $\Omega^*_{\mathcal{P}} = \mathsf{Büchi}(B^*)$ with

$$\begin{aligned} B^* = \big\{(v, P, W, w, p) \in V^*_{\mathcal{P}} \mid\ & (W = \{p\} \wedge w = 1) \vee && (1') \\ & (W = \emptyset \quad \wedge p \in P \wedge w = 1) \vee && (2') \\ & (W = \emptyset \quad \wedge \exists p' \in P,\ p < p')\big\}. && (3') \end{aligned}$$

The proof of the next proposition is a consequence of Theorem 4.

▶ **Proposition 5.** *Player 0 has a strategy σ_0 that is solution to the SPS problem in a reachability SP game $\mathcal{G}$ if and only if $\mathcal{P}$ has a winning strategy $\sigma^*_{\mathcal{P}}$ in $\mathcal{G}^*$.*

We obtain the following FPT algorithm for deciding the existence of a solution to the SPS problem in a reachability SP game $\mathcal{G}$. First, we construct the zero-sum game $\mathcal{G}^*$ whose number of vertices is linear in the number of vertices in the original game and double exponential in the number t of objectives of Player 1. Second, by Proposition 5, deciding whether there exists a solution to the SPS problem in $\mathcal{G}$ amounts to solving the zero-sum Büchi game $\mathcal{G}^*$; this can be done in quadratic time in the number of vertices of $\mathcal{G}^*$ [10]. Those two steps are in FPT for parameter t.

We now turn to parity SP games and briefly explain why solving the SPS problem in these games is in FPT, again by reduction to the C-P game. The arena G' of the C-P game remains as is and its objective $\Omega_{\mathcal{P}}$ is replaced by a *Boolean Büchi* objective $\Omega'_{\mathcal{P}}$ which encodes the three conditions for parity objectives. Boolean Büchi objectives are Boolean combinations of Büchi objectives and zero-sum games with such objectives are shown to be solvable in FPT in [8]. It follows that the SPS problem is also in FPT.

4 Complexity Class of the SPS Problem

In this section, we study the complexity class of the SPS problem and prove its NEXPTIME-completeness for both reachability and parity SP games.

4.1 NEXPTIME-Membership

We first show the membership to NEXPTIME of the SPS problem by providing a nondeterministic algorithm with time exponential in the size of the game $\mathcal{G}$. By *size*, we mean the number $|V|$ of its vertices and the number t of objectives of Player 1. Notice that the time complexity of the FPT algorithms obtained in the previous section is too high, preventing us from directly using the C-P game to show a tight membership result. Conversely, the nondeterministic algorithm provided in this section is not FPT as it is exponential in $|V|$.

▶ **Theorem 6.** *The SPS problem is in* NEXPTIME *for reachability and parity SP games.*

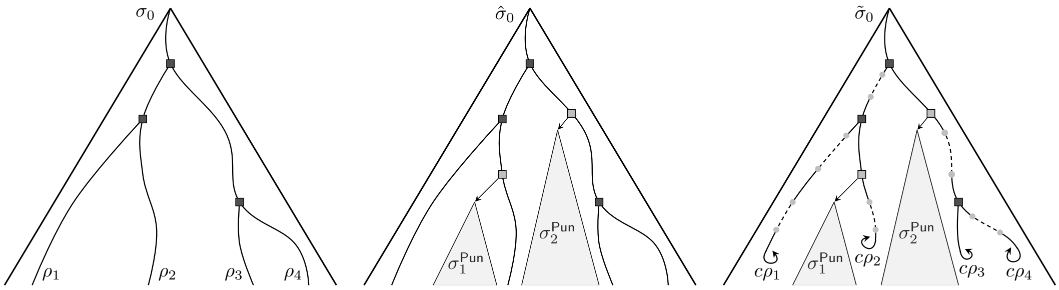


Figure 3 The creation of strategies $\hat{\sigma}_0$ and $\tilde{\sigma_0}$ from a solution σ_0 with $\mathsf{Wit}_{\sigma_0} = \{\rho_1, \rho_2, \rho_3, \rho_4\}$.

We show this membership result by proving that if Player 0 has a strategy which is a solution to the problem, he has one which is finite-memory with at most an exponential number of memory states[4]. This yields a NEXPTIME algorithm in which we nondeterministically guess such a strategy and check in exponential time that it is indeed a solution.

While our proof requires some specific arguments to treat both reachability and parity objectives, it is based on the following common principles. We first explain why, when there is a solution σ_0 to the SPS problem, there is one that is finite-memory. We consider a fixed set of witnesses Wit_{σ_0}. Figure 3 illustrates the two steps of the following construction.

- We start by showing the existence of a strategy $\hat{\sigma}_0$ constructed from σ_0, in which Player 0 follows σ_0 as long as the current consistent history is prefix of at least one witness in Wit_{σ_0}. Then when a deviation from Wit_{σ_0} occurs, Player 0 switches to a so-called *punishing strategy.* A deviation is a history that leaves the set of witnesses Wit_{σ_0} after a move of Player 1 (this is not possible by a move of Player 0). After such a deviation, $\hat{\sigma}_0$ systematically imposes that the consistent play either satisfies Ω_0 or is not σ_0-fixed Pareto-optimal, i.e., it gives to Player 1 a payoff that is strictly smaller than the payoff of a witness in Wit_{σ_0}. This makes the deviation *irrational* for Player 1. We show that this can be done, both for reachability and parity objectives, with at most exponentially many different punishing strategies, each having a size bounded exponentially in the size of the game. The strategy $\hat{\sigma}_0$ that we obtain is therefore composed of the part of σ_0 that produces Wit_{σ_0} and a punishment part whose size is at most exponential.
- Then, we show how to decompose each witness in Wit_{σ_0} into at most exponentially many *sections* that can, in turn, be compacted into finite elementary paths or lasso shaped paths of polynomial length. As Wit_{σ_0} contains exactly $|P_{\sigma_0}|$ witnesses ρ, those compact witnesses $c\rho$ can be produced by a finite-memory strategy with an exponential size for both reachability and parity objectives. This allows us to construct a strategy $\tilde{\sigma}_0$ that produces the compact witnesses and acts as $\hat{\sigma}_0$ after any deviation. This strategy is a solution of the SPS problem and has an exponential size as announced.

We can now sketch the proof of Theorem 6, again by giving arguments that work for both reachability and parity objectives. We guess a solution σ_0 to the SPS problem that we can assume to be finite-memory, that is, we guess it as a Moore machine $\mathcal{M}$ with an exponential number of memory states. We then verify that σ_0 is indeed a solution by first computing the set P_{σ_0} and then checking that every σ_0-fixed Pareto-optimal play satisfies the objective Ω_0 of Player 0. To this end, we construct the cartesian product $G \times \mathcal{M}$ which is an automaton whose infinite paths are exactly the plays consistent with σ_0. We then use classical results from automata theory about the emptiness problem for an intersection of reachability (resp. parity) objectives to get the announced exponential complexity of our verifying algorithm.

[4] Recall that to have a solution to the SPS problem, memory may be necessary as shown in Example 2.

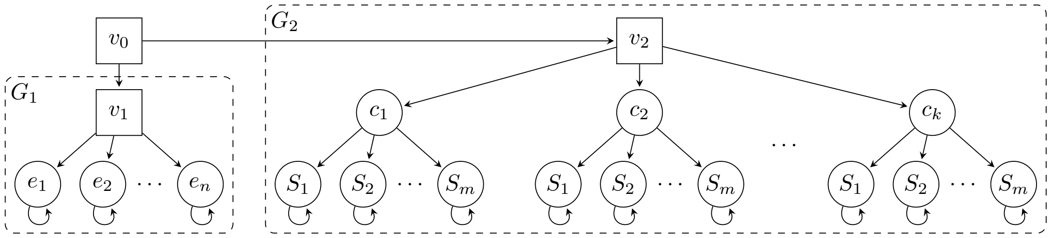


Figure 4 The tree arena used in the reduction from the SC problem.

4.2 NP-Completeness for Tree Arenas

Before turning to the NEXPTIME-hardness of the SPS problem in the next section, we first want to show that the SPS problem is already NP-complete in the simple setting of reachability objectives and arenas that are trees. To do so, we use a reduction from the Set Cover problem (SC problem) which is NP-complete [23].

▶ **Theorem 7.** *The SPS problem is* NP*-complete for reachability SP games on tree arenas.*

Notice that when the game arena is a tree, it is easy to design an algorithm for solving the SPS problem that is in NP. First, we nondeterministically guess a strategy σ_0 that can be assumed to be memoryless as the arena is a tree. Second, we apply a depth-first search algorithm from the root vertex which accumulates to leaf vertices the extended payoff of plays which are consistent with σ_0. Finally, we check that σ_0 is a solution.

Let us explain why the SPS problem is NP-hard on tree arenas by reduction from the SC problem. We recall that an instance of the SC problem is defined by a set $C = \{e_1, e_2, \dots, e_n\}$ of n elements, m subsets $S_1, S_2, \dots, S_m$ such that $S_i \subseteq C$ for each $i \in \{1, \dots, m\}$, and an integer $k \leq m$. The problem consists in finding k indexes $i_1, i_2, \dots, i_k$ such that the union of the corresponding subsets equals C, i.e., $C = \bigcup_{j=1}^{k} S_{i_j}$.

Given an instance of the SC problem, we construct a game with an arena consisting of $n + k \cdot (m + 1) + 3$ vertices. The arena G of the game is provided in Figure 4 and can be seen as two sub-arenas reachable from the initial vertex v_0. The game is such that there is a solution to the SC problem if and only if Player 0 has a strategy from v_0 in G which is a solution to the SPS problem. The game is played between Player 0 with reachability objective Ω_0 and Player 1 with $n + 1$ reachability objectives. The objectives are defined as follows: $\Omega_0 = \mathsf{Reach}(\{v_2\})$, $\Omega_i = \mathsf{Reach}(\{e_i\} \cup \{S_j \mid e_i \in S_j\})$ for $i \in \{1, 2, \dots, n\}$ and $\Omega_{n+1} = \mathsf{Reach}(\{v_2\})$. First, notice that every play in G_1 is consistent with any strategy of Player 0 and is lost by that player. It holds that for each $\ell \in \{1, 2, \dots, n\}$, there is such a play with payoff $(p_1, \dots, p_{n+1})$ such that $p_\ell = 1$ and $p_j = 0$ for $j \neq \ell$. These payoffs correspond to the elements e_ℓ which are to be covered in the SC problem. A play in G_2 visits v_2 and then a vertex c from which Player 0 selects a vertex S. Such a play is always won by Player 0 and its payoff is $(p_1, \dots, p_{n+1})$ such that $p_{n+1} = 1$ and $p_r = 1$ if and only if the element e_r belongs to the set S. It follows that the payoff of such a play corresponds to a set of elements in the SC problem. It is easy to see that the following proposition holds.

▶ **Proposition 8.** *There is a solution to an instance of the SC problem if and only if Player* 0 *has a strategy from* v_0 *in the corresponding SP game that is a solution to the SPS problem.*

4.3 NEXPTIME-Hardness

Let us come back to regular game arenas and show the NEXPTIME-hardness result thanks to the succinct variant of the SC problem presented below.

▶ **Theorem 9.** *The SPS problem is* NEXPTIME*-hard for reachability and parity SP games.*

Succinct Set Cover Problem. The *Succinct Set Cover problem (SSC problem)* is defined as follows. We are given a Conjunctive Normal Form (CNF) formula $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_p$ over the variables $X = \{x_1, x_2, \ldots, x_m\}$ made up of p clauses, each containing some disjunction of literals of the variables in X. The set of valuations of the variables X which satisfy ϕ is written $[\![\phi]\!]$. We are also given an integer $k \in \mathbb{N}$ (encoded in binary) and an other CNF formula $\psi = D_1 \wedge D_2 \wedge \cdots \wedge D_q$ over the variables $X \cup Y$ with $Y = \{y_1, y_2, \ldots, y_n\}$, made up of q clauses. Given a valuation $val_Y : Y \to \{0,1\}$ of the variables in Y, called a *partial valuation*, we write $\psi[val_Y]$ the CNF formula obtained by replacing in ψ each variable $y \in Y$ by its valuation $val_Y(y)$. We write $[\![\psi[val_Y]]\!]$ the valuations of the remaining variables X which satisfy $\psi[val_Y]$. The SSC problem is to decide whether there exists a set $K = \big\{val_Y \mid val_Y : Y \to \{0,1\}\big\}$ of k valuations of the variables in Y such that the valuations of the remaining variables X which satisfy the formulas $\psi[val_Y]$ include the valuations of X which satisfy ϕ. Formally, we write this $[\![\phi]\!] \subseteq \bigcup\limits_{val_Y \in K} [\![\psi[val_Y]]\!]$.

We can show that this corresponds to a set cover problem succinctly defined using CNF formulas. The set $[\![\phi]\!]$ of valuations of X which satisfy ϕ corresponds to the set of elements we aim to cover. Parameter k is the number of sets that can be used to cover these elements. Such a set is described by a formula $\psi[val_Y]$, given a partial valuation val_Y, and its elements are the valuations of X in $[\![\psi[val_Y]]\!]$. This is illustrated in the following example.

▶ **Example 10.** Consider the CNF formula $\phi = (x_1 \vee \neg x_2) \wedge (x_2 \vee x_3)$ over the variables $X = \{x_1, x_2, x_3\}$. The set of valuations of the variables which satisfy ϕ is $[\![\phi]\!] = \{(1,1,1), (1,1,0), (1,0,1), (0,0,1)\}$. Each such valuation corresponds to one element we aim to cover. Consider the CNF formula $\psi = (y_1 \vee y_2) \wedge (x_1 \vee y_2) \wedge (x_2 \vee x_3 \vee y_1)$ over the variables $X \cup Y$ with $Y = \{y_1, y_2\}$. Given the partial valuation val_Y of the variables in Y such that $val_Y(y_1) = 0$ and $val_Y(y_2) = 1$, we get the CNF formula $\psi[val_Y] = (0 \vee 1) \wedge (x_1 \vee 1) \wedge (x_2 \vee x_3 \vee 0)$. This formula describes the contents of the set identified by the partial valuation (as a partial valuation yields a unique formula). The valuations of the variables X which satisfy $\psi[val_Y]$ are the elements contained in the set. In this case, these elements are $[\![\psi[val_Y]]\!] = \{(0,1,0), (0,0,1), (0,1,1), (1,1,0), (1,0,1), (1,1,1)\}$. We can see that this set contains the elements $\{(1,1,1), (1,1,0), (1,0,1), (0,0,1)\}$ of $[\![\phi]\!]$.

It is easy to see that the SSC problem is in NEXPTIME. Its NEXPTIME-hardness can be obtained by reduction from the Succinct Dominating Set problem, which is NEXPTIME-complete for graphs succinctly encoded using CNF formulas [14].

▶ **Theorem 11.** *The SSC problem is* NEXPTIME*-complete.*

We now describe our reduction from the SSC problem to show the NEXPTIME-hardness of solving the SPS problem for reachability SP games. The proof of this result for parity SP games uses similar arguments, adapted to the prefix-independent nature of parity objectives.

Given an instance of the SSC problem, we construct a reachability SP game with arena G consisting of a polynomial number of vertices in the number of clauses and variables in the formulas ϕ and ψ and in the length of the binary encoding of the integer k. This reduction is such that there is a solution to the SSC problem if and only if Player 0 has a strategy

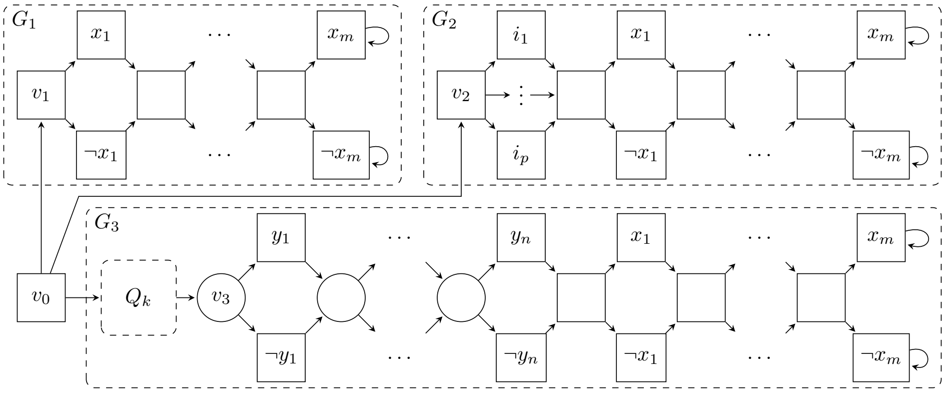


Figure 5 The arena G used in the reduction from the SSC problem.

from v_0 in G which is a solution to the SPS problem. The arena G, provided in Figure 5, can be viewed as three sub-arenas reachable from v_0. Sub-arenas G_1 and G_2 are completely controlled by Player 1. Plays entering these sub-arenas are therefore consistent with any strategy of Player 0. Sub-arena G_3 starts with a gadget Q_k whose vertices belong to Player 1 and which provides exactly k different paths from v_0 to v_3.

Objectives. The game is played between Player 0 with reachability objective Ω_0 and Player 1 with $t = 1 + 2 \cdot m + p + q$ reachability objectives. The payoff of a play therefore consists in a single Boolean for objective Ω_1, a vector of $2 \cdot m$ Booleans for objectives $\Omega_{x_1}, \Omega_{\neg x_1}, \dots, \Omega_{x_m}, \Omega_{\neg x_m}$, a vector of p Booleans for objectives $\Omega_{C_1}, \dots, \Omega_{C_p}$ and a vector of q Booleans for objectives $\Omega_{D_1}, \dots, \Omega_{D_q}$. The objectives are defined as follows.

- The target set for objective Ω_0 of Player 0 and objective Ω_1 of Player 1 is $\{v_2, v_3\}$.
- The target set for objective Ω_{x_i} (resp. $\Omega_{\neg x_i}$) with $i \in \{1, \dots, m\}$ is the set of vertices labeled x_i (resp. $\neg x_i$) in G_1, G_2 and G_3.
- The target set for objective Ω_{C_i} with $i \in \{1, \dots, p\}$ is the set of vertices in G_1 and G_3 corresponding to the literals of X which make up the clause C_i in ϕ. In addition, vertex i_j in G_2 belongs to the target set of objective Ω_{C_ℓ} for all $\ell \in \{1, \dots, p\}$ such that $\ell \neq j$.
- The target set of objective Ω_{D_i} with $i \in \{1, \dots, q\}$ is the set of vertices in G_3 corresponding to the literals of X and Y which make up the clause D_i in ψ. In addition, vertices v_1 and v_2 satisfy every objective Ω_{D_i} with $i \in \{1, \dots, q\}$.

Payoff of Plays in G_1. Plays in G_1 do not satisfy objective Ω_0 of Player 0 nor objective Ω_1 of Player 1. A play in G_1 is of the form $v_0\, v_1\, z_1 \,\square \cdots \square\, (z_m)^\omega$ where z_i is either x_i or $\neg x_i$. It follows that a play satisfies the objective Ω_{x_i} or $\Omega_{\neg x_i}$ for each $x_i \in X$. The vector of payoffs for these objectives corresponds to a valuation of the variables in X, expressed as a vector of $2 \cdot m$ Booleans. In addition, due to the way the objectives are defined, objective Ω_{C_i} is satisfied in a play if and only if clause C_i of ϕ is satisfied by the valuation this play corresponds to. The objective Ω_{D_i} for $i \in \{1, \dots, q\}$ is satisfied in every play in G_1.

▶ **Lemma 12.** *Plays in G_1 are consistent with any strategy of Player 0. Their payoff are of the form $(0, val, sat(\phi, val), 1, \dots, 1)$ where val is a valuation of the variables in X expressed as a vector of payoffs for objectives Ω_{x_1} to $\Omega_{\neg x_m}$ and $sat(\phi, val)$ is the vector of payoffs for objectives Ω_{C_1} to Ω_{C_p} corresponding to that valuation. All plays in G_1 are lost by Player 0.*

Payoff of Plays in G_2. Plays in G_2 satisfy the objectives Ω_0 of Player 0 and Ω_1 of Player 1. A play in G_2 is of the form $v_0\, v_2\, i_j \,\square\, z_1 \,\square \cdots \square\, (z_m)^\omega$ where z_ℓ is either x_ℓ or $\neg x_\ell$. It follows that a play satisfies either the objective Ω_x or $\Omega_{\neg x}$ for each $x \in X$ which again corresponds to a valuation of the variables in X. The objective Ω_{D_i} for $i \in \{1, \dots, q\}$ is satisfied in every play in G_2. Compared to the plays in G_1, the difference lies in the objectives corresponding to clauses of ϕ which are satisfied. In any play in G_2, a vertex i_j with $j \in \{1, \dots, p\}$ is first visited, satisfying all the objectives Ω_{C_ℓ} with $\ell \in \{1, \dots, p\}$ and $\ell \neq j$. All but one objective corresponding to the clauses of ϕ are therefore satisfied.

▶ **Lemma 13.** *Plays in G_2 are consistent with any strategy of Player 0. Their payoff are of the form $(1, val, vec, 1, \dots, 1)$ where val is a valuation of the variables in X expressed as a vector of payoffs for objectives Ω_{x_1} to $\Omega_{\neg x_m}$ and vec is a vector of payoffs for objectives Ω_{C_1} to Ω_{C_p} in which all of them except one are satisfied. All plays in G_2 are won by Player 0.*

Plays in G_2 are such that their payoff is strictly larger than the payoff of plays in G_1 whose valuation of X does not satisfy ϕ. It is easy to see that, when considering G_1 and G_2, the only plays in G_1 with a Pareto-optimal payoff are exactly those whose valuation satisfies all clauses of ϕ. The following lemma therefore holds.

▶ **Lemma 14.** *The set of payoffs of plays in G_1 that are σ_0-fixed Pareto-optimal when considering $G_1 \cup G_2$ for any strategy σ_0 of Player 0 is equal to the set of payoffs of plays in G_1 whose valuation of X satisfy ϕ.*

Problematic Payoffs in G_1. The plays described in the previous lemma correspond exactly to the valuations of X which satisfy ϕ and therefore to the elements we aim to cover in the SSC problem. They are σ_0-fixed Pareto-optimal when considering $G_1 \cup G_2$ and are lost by Player 0. All other σ_0-fixed Pareto-optimal payoffs in $G_1 \cup G_2$ are only realized by plays in G_2 which are all won by Player 0. It follows that in order for Player 0 to find a strategy σ_0 from v_0 that is solution to the SPS problem, it must hold that those payoffs are not σ_0-fixed Pareto-optimal when considering $G_1 \cup G_2 \cup G_3$. Otherwise, a play consistent with σ_0 with a σ_0-fixed Pareto-optimal payoff is lost by Player 0. We call those payoffs *problematic payoffs.*

In order for Player 0 to find a strategy σ_0 which is a solution to the SPS problem, this strategy must be such that for each problematic payoff in G_1, there is a play in G_3 consistent with σ_0 and with a strictly larger payoff. Since the plays in G_3 are all won by Player 0, this would ensure that the strategy σ_0 is a solution to the problem. This corresponds in the SSC problem to selecting a series of sets in order to cover the valuations of X which satisfy ϕ.

Payoff of Plays in G_3. Plays in G_3 satisfy the objectives Ω_0 of Player 0 and Ω_1 of Player 1. A play in G_3 consistent with a strategy σ_0 is of the form $v_0 \square \cdots \square v_3\, r_1 \bigcirc \cdots \bigcirc r_n \square z_1 \square \cdots \square (z_m)^\omega$ where r_i is either y_i or $\neg y_i$ and z_i is either x_i or $\neg x_i$. Since only the vertices leading to y or $\neg y$ for $y \in Y$ belong to Player 0, it holds that $v_3\, r_1 \bigcirc \cdots \bigcirc r_n$ is the only part of any play in G_3 which is directly influenced by σ_0. That part of a play comes after a history from v_0 to v_3 of which there are k, provided by gadget Q_k. By definition of a strategy, this can be interpreted as Player 0 making a choice of valuation of the variables in Y after each of those k histories. After this, the play satisfies either the objective Ω_x or $\Omega_{\neg x}$ for each $x \in X$ which corresponds to a valuation of X. Due to the way the objectives are defined, the objective Ω_{C_i} (resp. Ω_{D_i}) is satisfied if and only if clause C_i of ϕ (resp. D_i of ψ) is satisfied by the valuation of the variables in X (resp. X and Y) the play corresponds to.

Creating Strictly Larger Payoffs in G_3. In order to create a play with a payoff r' that is strictly larger than a problematic payoff r, σ_0 must choose a valuation of Y such that there exists a valuation of the remaining variables X which together with this valuation of Y satisfies ψ and ϕ (since in r every objective Ω_{C_i} for $i \in \{1, \dots, p\}$ and Ω_{D_i} for $i \in \{1, \dots, q\}$ is satisfied). Since the plays in G_3 also satisfy the objective Ω_1 and plays in G_1 do not, this ensures that $r < r'$.

We finally briefly explain why the proposed reduction is correct. In case of a positive instance of the SSC problem, by carefully selecting k valuations of Y, Player 0 ensures that for each valuation val_X satisfying ϕ, there is a play in G_3 with a valuation val_Y such that $val_X \in [\![\psi[val_Y]]\!]$. Therefore, when considering the whole arena, no play in G_1 is Pareto-optimal and every Pareto-optimal play is won by Player 0. In case of a negative instance, Player 0 is not able to do so and some play in G_1 thus has a Pareto-optimal payoff and is lost by Player 0.

5 Conclusion

We have introduced in this paper the class of two-player SP games and the SPS problem in those games. We provided a reduction from SP games to a two-player zero-sum game called the C-P game, which we used to obtain FPT results on solving this problem. We showed how the arena and the generic objective of this C-P game can be adapted to specifically handle reachability and parity SP games. This allowed us to prove that reachability (resp. parity) SP games are in FPT when the number t of objectives of Player 1 (resp. when t and the maximal priority according to each priority function in the game) is a parameter. We then turned to the complexity class of the SPS problem and sketched the main arguments used in our proof of its NEXPTIME-membership, which relied on showing that any solution to the SPS problem in a reachability or parity SP game can be transformed into a solution with an exponential memory. We showed the NP-completeness of the problem in the simple setting of reachability SP games played on tree arenas. We then came back to regular game arenas and established the NEXPTIME-hardness of the SPS problem in reachability and parity SP games. This result relied on a reduction from the SSC problem which we proved to be NEXPTIME-complete, a result of potential independent interest.

In future work, we want to study other ω-regular objectives as well as quantitative objectives such as mean-payoff in the framework of SP games and the SPS problem. It would also be interesting to study whether other works, such as rational synthesis, could benefit from the approaches used in this paper.

References

1 Mrudula Balachander, Shibashis Guha, and Jean-François Raskin. Fragility and robustness in mean-payoff adversarial Stackelberg games. *CoRR*, abs/2007.07209, 2020. arXiv:2007.07209.

2 Dietmar Berwanger. Admissibility in infinite games. In Wolfgang Thomas and Pascal Weil, editors, *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science, Aachen, Germany, February 22-24, 2007, Proceedings*, volume 4393 of *Lecture Notes in Computer Science*, pages 188–199. Springer, 2007. doi:10.1007/978-3-540-70918-3_17.

3 Romain Brenguier, Lorenzo Clemente, Paul Hunter, Guillermo A. Pérez, Mickael Randour, Jean-François Raskin, Ocan Sankur, and Mathieu Sassolas. Non-zero sum games for reactive synthesis. In Adrian-Horia Dediu, Jan Janousek, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications – 10th International Conference, LATA 2016, Prague, Czech Republic, March 14-18, 2016, Proceedings*, volume 9618 of *Lecture Notes in Computer Science*, pages 3–23. Springer, 2016. doi:10.1007/978-3-319-30000-9_1.

4 Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. In Luca Aceto and David de Frutos-Escrig, editors, *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015*, volume 42 of *LIPIcs*, pages 100–113. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.CONCUR.2015.100.

5 Léonard Brice, Jean-François Raskin, and Marie van den Bogaard. Subgame-perfect equilibria in mean-payoff games. *CoRR*, abs/2101.10685, 2021. arXiv:2101.10685.

6 Thomas Brihaye, Véronique Bruyère, Aline Goeminne, Jean-François Raskin, and Marie van den Bogaard. The complexity of subgame perfect equilibria in quantitative reachability games. *Log. Methods Comput. Sci.*, 16(4), 2020. URL: https://lmcs.episciences.org/6883.

7 Véronique Bruyère. Computer aided synthesis: A game-theoretic approach. In Émilie Charlier, Julien Leroy, and Michel Rigo, editors, *Developments in Language Theory – 21st International Conference, DLT 2017, Liège, Belgium, August 7-11, 2017, Proceedings*, volume 10396 of *Lecture Notes in Computer Science*, pages 3–35. Springer, 2017. doi:10.1007/978-3-319-62809-7_1.

8 Véronique Bruyère, Quentin Hautem, and Jean-François Raskin. Parameterized complexity of games with monotonically ordered omega-regular objectives. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.29.

9 Véronique Bruyère, Jean-François Raskin, and Clément Tamines. Stackelberg-Pareto synthesis (full version). *CoRR*, abs/2102.08925, 2021. arXiv:2102.08925.

10 Krishnendu Chatterjee and Monika Henzinger. Efficient and dynamic algorithms for alternating Büchi games and maximal end-component decomposition. *J. ACM*, 61(3):15:1–15:40, 2014. doi:10.1145/2597631.

11 Krishnendu Chatterjee and Thomas A. Henzinger. Assume-guarantee synthesis. In Orna Grumberg and Michael Huth, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24 – April 1, 2007, Proceedings*, volume 4424 of *Lecture Notes in Computer Science*, pages 261–275. Springer, 2007. doi:10.1007/978-3-540-71209-1_21.

12 Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdzinski. Games with secure equilibria. *Theor. Comput. Sci.*, 365(1-2):67–82, 2006. doi:10.1016/j.tcs.2006.07.032.

13 Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The complexity of rational synthesis. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, volume 55 of *LIPIcs*, pages 121:1–121:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.ICALP.2016.121.

14 Bireswar Das, Patrick Scharpfenecker, and Jacobo Torán. CNF and DNF succinct graph encodings. *Inf. Comput.*, 253:436–447, 2017. doi:10.1016/j.ic.2016.06.009.

15 R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer New York, 2012. URL: https://books.google.be/books?id=HyTjBwAAQBAJ.

16 Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The adversarial Stackelberg value in quantitative games. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8–11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 127:1–127:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.127.

17 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational synthesis. In Javier Esparza and Rupak Majumdar, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6015 of *Lecture Notes in Computer Science*, pages 190–204. Springer, 2010. doi:10.1007/978-3-642-12002-2_16.

18 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-36387-4.

19 Anshul Gupta and Sven Schewe. Quantitative verification in rational environments. In Amedeo Cesta, Carlo Combi, and François Laroussinie, editors, *21st International Symposium on Temporal Representation and Reasoning, TIME 2014, Verona, Italy, September 8-10, 2014*, pages 123–131. IEEE Computer Society, 2014. doi:10.1109/TIME.2014.9.

20 Anshul Gupta, Sven Schewe, and Dominik Wojtczak. Making the best of limited memory in multi-player discounted sum games. In Javier Esparza and Enrico Tronci, editors, *Proceedings Sixth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2015, Genoa, Italy, 21-22nd September 2015*, volume 193 of *EPTCS*, pages 16–30, 2015. doi:10.4204/EPTCS.193.2.

21 Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael J. Wooldridge. On computational tractability for rational verification. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 329–335. ijcai.org, 2019. doi:10.24963/ijcai.2019/47.

22 Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael J. Wooldridge. Automated temporal equilibrium analysis: Verification and synthesis of multi-player games. *Artif. Intell.*, 287:103353, 2020. doi:10.1016/j.artint.2020.103353.

23 Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

24 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with rational environments. *Ann. Math. Artif. Intell.*, 78(1):3–20, 2016. doi:10.1007/s10472-016-9508-8.

25 John F. Nash. Equilibrium points in n-person games. In *PNAS*, volume 36, pages 48–49. National Academy of Sciences, 1950.

26 Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pages 179–190. ACM Press, 1989. doi:10.1145/75277.75293.

27 Reinhard Selten. Spieltheoretische Behandlung eines Oligopolmodells mit Nachfrageträgheit. *Zeitschrift für die gesamte Staatswissenschaft*, 121:301–324 and 667–689, 1965.

28 Michael Ummels. Rational behaviour and strategy construction in infinite multiplayer games. In S. Arun-Kumar and Naveen Garg, editors, *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science, 26th International Conference, Kolkata, India, December 13-15, 2006, Proceedings*, volume 4337 of *Lecture Notes in Computer Science*, pages 212–223. Springer, 2006. doi:10.1007/11944836_21.

29 Michael Ummels. The complexity of Nash equilibria in infinite multiplayer games. In Roberto M. Amadio, editor, *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29 – April 6, 2008. Proceedings*, volume 4962 of *Lecture Notes in Computer Science*, pages 20–34. Springer, 2008. doi:10.1007/978-3-540-78499-9_3.

30 Michael Ummels and Dominik Wojtczak. The complexity of Nash equilibria in limit-average games. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR 2011 – Concurrency Theory – 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*, volume 6901 of *Lecture Notes in Computer Science*, pages 482–496. Springer, 2011. doi:10.1007/978-3-642-23217-6_32.

31 Heinrich Freiherr von Stackelberg. *Marktform und Gleichgewicht*. Wien und Berlin, J. Springer, Cambridge, MA, 1937.

The Orbit Problem for Parametric Linear Dynamical Systems

Christel Baier
Technische Universität Dresden, Germany

Florian Funke
Technische Universität Dresden, Germany

Simon Jantsch
Technische Universität Dresden, Germany

Toghrul Karimov
Max Planck Institute for Software Systems,
Saarland Informatics Campus,
Saarbrücken, Germany

Engel Lefaucheux
Max Planck Institute for Software Systems,
Saarland Informatics Campus,
Saarbrücken, Germany

Florian Luca
School of Mathematics, Wits University,
Johannesburg, South Africa
Research Group in Algebraic Structures & Applications, King Abdulaziz University,
Thuwal, Saudi Arabia
Max Planck Institute for Software Systems,
Saarland Informatics Campus,
Saarbrücken, Germany

Joël Ouaknine
Max Planck Institute for Software Systems,
Saarland Informatics Campus,
Saarbrücken, Germany

David Purser
Max Planck Institute for Software Systems,
Saarland Informatics Campus,
Saarbrücken, Germany

Markus A. Whiteland
Max Planck Institute for Software Systems,
Saarland Informatics Campus,
Saarbrücken, Germany

James Worrell
Department of Computer Science,
University of Oxford, UK

Abstract

We study a parametric version of the Kannan-Lipton Orbit Problem for linear dynamical systems. We show decidability in the case of one parameter and Skolem-hardness with two or more parameters.

More precisely, consider a d-dimensional square matrix M whose entries are algebraic functions in one or more real variables. Given initial and target vectors $u, v \in \mathbb{Q}^d$, the parametric point-to-point orbit problem asks whether there exist values of the parameters giving rise to a concrete matrix $N \in \mathbb{R}^{d \times d}$, and a positive integer $n \in \mathbb{N}$, such that $N^n u = v$.

We show decidability for the case in which M depends only upon a single parameter, and we exhibit a reduction from the well-known Skolem Problem for linear recurrence sequences, suggesting intractability in the case of two or more parameters.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Orbit problem, parametric, linear dynamical systems

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.28

Related Version *Full Version (with full proofs)*: https://arxiv.org/abs/2104.10634

Funding This work was funded by DFG grant 389792660 as part of TRR 248 – CPEC (see perspicuous-computing.science), the Cluster of Excellence EXC 2050/1 (CeTI, project ID 390696704, as part of Germany's Excellence Strategy), DFG-projects BA-1679/11-1 and BA-1679/12-1, and the Research Training Group QuantLA (GRK 1763).
Joël Ouaknine: ERC grant AVS-ISS (648701). Also affiliated with Keble College, Oxford as emmy.network Fellow.
James Worrell: Supported by EPSRC Fellowship EP/N008197/1.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 28; pp. 28:1–28:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

The *Orbit Problem* for linear dynamical systems asks to decide, given a square matrix $M \in \mathbb{Q}^{d\times d}$ and two vectors $u, v \in \mathbb{Q}^d$, whether there exists a natural number n such that $M^n u = v$. The problem was shown decidable (in polynomial time) by Kannan and Lipton [26] over ten years after Harrison first raised the question of decidability [23]. The current paper is concerned with a generalisation of the Orbit Problem to *parametric* linear dynamical systems. In general, parametric models address a major drawback in quantitative verification, namely the unrealistic assumption that quantitative data in models are known *a priori* and can be specified exactly. In applications of linear dynamical systems to automated verification, parameters are used to model partially specified systems (e.g., a faulty component with an unknown failure rate, or when transition probabilities are only known up to some bounded precision) as well as to model the unknown environment of a system. Interval Markov chains can also be considered as a type of parametric linear dynamical system.

▶ **Problem 1** (Parametric Orbit Problem)**.** Given a $(d \times d)$-matrix M, initial and target vectors u, v, whose entries are real algebraic functions in ℓ common real variables $X = (x_1, ..., x_\ell)$, does there exist $s \in \mathbb{R}^\ell$, i.e., values of the parameters giving rise to a concrete matrix, initial and target $M(s) \in \mathbb{R}^{d\times d}, u(s), u(s) \in \mathbb{R}^d$, and a positive integer $n \in \mathbb{N}$, such that $M(s)^n u(s) = v(s)$?

We prove two main results in this paper. In the case of a single parameter we show that the Parametric Orbit Problem is decidable. On the other hand, we show that the Parametric Orbit Problem is at least as hard as the Skolem Problem – a well-known decision problem for linear recurrence sequences, whose decidability has remained open for many decades. Our reduction establishes intractability in the case of two or more parameters.

Thus our main decidability result is as follows:

▶ **Theorem 2.** *Problem 1 is decidable when there is a single parameter (i.e., $\ell = 1$).*

Theorem 2 concerns a reachability problem in which the parameters are existentially quantified. It would be straightforward to adapt our methods to allow additional constraints on the parameter, e.g., requiring that s lie in a certain specified interval. In terms of verification, a negative answer to an instance of the above reachability problem could be seen as establishing a form of robust safety, i.e., an “error state” is not reachable regardless of the value of the unknown parameter.

The proof of Theorem 2 follows a case distinction based on properties of the eigenvectors of the matrix M (whose entries are functions) and the shape of the Jordan normal form J of M. Our theorem assumes the entries of the matrix, initial and target vectors are real algebraic functions – in particular encompassing polynomial and rational functions. Note that even if we were to restrict the entries of M to be polynomials in the parameters, we would still require (complex) algebraic functions in the Jordan normal form. We assume a suitable effective representation of algebraic functions that supports evaluation at algebraic points, computing the range and zeros of the functions, arithmetic operations, and extracting roots of polynomials whose coefficients are algebraic functions.

The most challenging cases arise when J is diagonal. In this situation we can reformulate the problem as follows: given algebraic functions $\lambda_i(x), \gamma_i(x)$ for $1 \le i \le t$, does there exist $(n, s) \in \mathbb{N} \times \mathbb{R}$ such that

$$\lambda_i^n(s) = \gamma_i(s) \qquad \text{for all} \qquad i = 1, \ldots, t? \tag{1}$$

A further key distinction in analysing the problem in Equation (1) involves the rank of the multiplicative group generated by the functions $\lambda_1, \ldots, \lambda_t$. To handle the case that the group has rank at least two, a central role is played by the results of Bombieri, Masser, and Zannier (see [8, Theorem 2] and [9]) concerning the intersection of a curve in $\mathbb{C}^m$, with algebraic subgroups of $(\mathbb{C}^*)^m$ of dimension at most $m-2$. To apply these results we view the problem in Equation (1) geometrically in terms of whether a curve

$$C = \{(\lambda_1(s), \ldots, \lambda_t(s), \gamma_1(s), \ldots, \gamma_t(s)) : s \in \mathbb{R}\} \subseteq \mathbb{C}^{2t}$$

intersects the multiplicative group

$$G_n = \{(\alpha_1, \ldots, \alpha_t, \beta_1, \ldots, \beta_t) \in (\mathbb{C}^*)^{2t} \colon \alpha_1^n = \beta_1 \wedge \cdots \wedge \alpha_t^n = \beta_t\}$$

for some $n \in \mathbb{N}$. The above-mentioned results of Bombieri, Masser, and Zannier can be used to derive an upper bound on n such that $C \cap G_n$ is non-empty under certain conditions on the set of multiplicative relations holding among $\lambda_1, \ldots, \lambda_t$ and $\gamma_1, \ldots, \gamma_t$.

We provide specialised arguments for a number of cases for which the results of Bombieri, Masser, and Zannier cannot be applied. In particular, for the case that the multiplicative group generated by the functions $\lambda_1, \ldots, \lambda_t$ has rank one, we provide in Section 6 a direct elementary method to find solutions of Equation (1).

Another main case in the proof is when matrix J has a Jordan block of size at least 2, i.e., it is not diagonal (see Section 4.2). The key instrument here is the notion of the Weil height of an algebraic number together with bounds that relate the height of a number to the height of its image under an algebraic function. Using these bounds we obtain an upper bound on the $n \in \mathbb{N}$ such that the equation $M(s)^n u(s) = v(s)$ admits a solution $s \in \mathbb{R}$.

Related work

Reachability problems in (unparametrized) linear dynamical systems have a rich history. Answering a question by Harrison [23], Kannan and Lipton [26] showed that the point-to-point reachability problem in linear dynamical systems is decidable in PTIME. They also noticed that the problem becomes significantly harder if the target is a linear subspace – a problem that still remains open, but has been solved for low-dimensional instances [14]. This was extended to polytope targets in [15], and later further generalized to polytope initial sets in [2]. Orbit problems have recently been studied in the setting of rounding functions [3]. In our analysis we will make use of a version of the point-to-point reachability problem that allows matrix entries to be algebraic numbers. In this case the eigenvalues are again algebraic, and decidability follows by exactly the same argument as the rational case (although the algorithm is no longer in PTIME), and is also a special case of the main result of [10].

If the parametric matrix M is the transition matrix of a parametric Markov chain (pMC) [24, 22, 28], then our approach combines *parameter synthesis* with the *distribution transformer semantics*. Parameter synthesis on pMCs asks whether some (or every) parameter setting results in a Markov chain satisfying a given specification, expressed, e.g., in PCTL [25]. An important problem in this direction is to find parameter settings with prescribed properties [30, 12, 19], which has also been studied in the context of model repair [4, 37]. While all previous references use the standard path-based semantics of Markov chains, the distribution transformer semantics [29, 27, 13] studies the transition behaviour on probability distributions. It has, to the best of our knowledge, never been considered for parametric Markov chains. Our approach implicitly does this in that it performs parameter synthesis for a reachability property in the distribution transformer semantics.

The Skolem Problem asks whether a linear recurrence sequence $(u_n)_n$ has a zero term (n such that $u_n = 0$). Phrased in terms of linear dynamical systems, the Skolem Problem asks whether a d-dimensional linear dynamical system hits a $(d-1)$-dimensional hyperplane, and decidability in this setting is known for matrices of dimension at most four [34, 39]. A continuous version of the Skolem Problem was examined in [16]. With the longstanding intractability of the Skolem Problem in general, it has recently been used as a reference point for other decision problems [1, 32, 38].

Ostafe and Shparlinski [35] consider the Skolem Problem for parametric families of simple linear recurrences. More precisely, they consider linear recurrences of the form $u_n = a_1(x)\lambda_1(x)^n + \cdots + a_k(x)\lambda_k^n(x)$ for rational functions $a_1, \ldots, a_k, \lambda_1, \ldots, \lambda_k$ with coefficients in a number field. They show that the existence of a zero of the sequence (u_n) can be decided for all values of the parameter outside an exceptional set of numbers of bounded height (note that any value of the parameter such that the sequence u_n has a zero is necessarily algebraic).

2 Preliminaries

We denote by $\mathbb{R}, \mathbb{C}, \mathbb{Q}, \overline{\mathbb{Q}}$ the real, complex, rational, and algebraic numbers respectively. For a field K and a finite set X of variables, $K[X]$ and $K(X)$ respectively denote the ring of polynomials and field of rational functions with coefficients in K. A meromorphic function[1] $f\colon U \to \mathbb{C}$ where U is some open subset $U \subseteq \mathbb{C}^\ell$ is called algebraic, if $P(x_1, \ldots, x_\ell, f(x_1, \ldots, x_\ell)) = 0$ for some $P \in \mathbb{Q}[x_1, \ldots, x_\ell, y]$. We say that f is *real algebraic* if it is real-valued on real inputs.

▶ **Definition 3.** *A* parametric Linear Dynamical System *(pLDS) of* dimension $d \in \mathbb{N}$ *is a tuple $\mathcal{M} = (X, M, u)$, where X is a finite set of* parameters*, M is the* parametrized matrix *whose entries are real algebraic functions in parameters X and u is the parametric initial distribution whose entries are also real algebraic functions in parameters X.*

Given $s \in \mathbb{R}^{|X|}$, we denote by $M(s)$ the matrix $\mathbb{R}^{d\times d}$ obtained from M by evaluating each function in M at s, provided that this value is well-defined. Likewise we obtain $u(s)$. We call $(M(s), u(s))$ the induced linear dynamical system (LDS). The *orbit* of the LDS $(M(s), u(s))$ is the set of vectors obtained by repeatedly applying the matrix $M(s)$ to $u(s)$: $\{u(s), M(s)u(s), M(s)^2u(s), \ldots\}$. The LDS $(M(s), u(s))$ *reaches* a target $v(s)$ if $v(s)$ is in the orbit, *i.e.* there exists $n \in \mathbb{N}$ such that $M(s)^n u(s) = v(s)$.

We remark that $M(s)$ is undefined whenever any of the entries of M is undefined. For any fixed n, the elements of M^n are polynomials in the entries of M, and consequently, M^n is defined on the same domain as M.

Unless we state that M is a constant function, all matrices should be seen as functions, with parameters $x_1, \ldots, x_{|X|}$, or simply x if there is a single parameter. The notation s is used for a specific instantiation of x. We often omit x when referring to a function, either the function is declared constant or when we do not need to make reference to its parameters.

2.1 Computation with algebraic numbers

Throughout this note we employ notions from (computational) algebraic geometry and algebraic number theory. Our approach relies on transforming the matrices we consider in Jordan normal form. Doing so, the coefficients of the computed matrix are not rational anymore but algebraic. Next we recall the necessary basics and refer to [17, 40] for more background on notions utilised throughout the text.

[1] A ratio of two holomorphic functions, which are complex-valued functions complex differentiable in some neighbourhood of every point of the domain.

The algebraic numbers $\overline{\mathbb{Q}}$ are the complex numbers which can be defined as some root of a univariate polynomial in $\mathbb{Q}[x]$. In particular, the rational numbers are algebraic numbers. For every $\alpha \in \overline{\mathbb{Q}}$ there exists a unique monic univariate polynomial $P_\alpha \in \mathbb{Q}[x]$ of minimum degree for which $P_\alpha(\alpha) = 0$. We call P_α the *minimal polynomial* of α. An algebraic number α is represented as a tuple $(P_\alpha, \alpha^*, \varepsilon)$, where $\alpha^* = a_1 + a_2 i$, $a_1, a_2 \in \mathbb{Q}$, is an approximation of α, and $\varepsilon \in \mathbb{Q}$ is sufficiently small such that α is the unique root of P_α within distance ε of α^* (such ε can be computed by the root-separation bound, due to Mignotte [33]). This is referred to as the *standard* or *canonical representation* of an algebraic number. Given canonical representations of two algebraic numbers α and β, one can compute canonical representations of $\alpha + \beta$, $\alpha\beta$, and α/β, all in polynomial time.

▶ **Definition 4** (Weil's absolute logarithmic height)**.** *Given an algebraic number α with minimal polynomial p_α of degree d, consider the polynomial $a_d p_\alpha$ with $a_d \in \mathbb{N}$ minimal such that for $a_d p_\alpha = a_d x^d + \cdots + a_1 x + a_0$ we have $a_i \in \mathbb{Z}$ and $\gcd(a_1, \ldots, a_d) = 1$. Write $a_d p_\alpha = a_d(x - \alpha^{(1)}) \cdots (x - \alpha^{(d)})$, where $\alpha^{(1)} = \alpha$. Define the* (Weil) height *$h(\alpha)$ of $\alpha \neq 0$ by $h(\alpha) = \frac{1}{d}\Big(\log a_d + \sum_{i=1}^{d} \log(\max\{|\alpha^{(i)}|, 1\})\Big)$. By convention $h(0) = 0$.*

For all $\alpha, \beta \in \overline{\mathbb{Q}}$ and $n \in \mathbb{Z}$ we have from [40, Chapt. 3]:
1. $h(\alpha + \beta) \leq h(\alpha) + h(\beta) + \log 2$;
2. $h(\alpha\beta) \leq h(\alpha) + h(\beta)$;
3. $h(\alpha^n) = |n| \cdot h(\alpha)$.

In addition, for $\alpha \neq 0$ we have $h(\alpha) = 0$ if and only if α is a root of unity (α is a root of unity if there exists $k \in \mathbb{N}$, $k \geq 1$, such that $\alpha^k = 1$). Notice that the set of algebraic numbers with both height and degree bounded is always finite.

2.2 Univariate algebraic functions

Let K be an algebraic extension of a field L such that the characteristic polynomial of $M \in L^{d \times d}$ splits into linear factors over K. It is well-known that we can factor M over K as $M = C^{-1}JC$ for some invertible matrix $C \in K^{d\times d}$ and block diagonal Jordan matrix $J = \langle J_1, \ldots, J_N\rangle \in K^{d \times d}$. Each block J_i associated with some eigenvalue λ_i, and J_i^n, have the following *Jordan block* form for some $k \geq 1$:

$$J_i = \begin{pmatrix} \lambda & 1 & 0 & \cdots & 0 \\ 0 & \lambda & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ 0 & 0 & 0 & \cdots & \lambda \end{pmatrix} \qquad \text{and} \qquad J_i^n = \begin{pmatrix} \lambda^n & n\lambda^{n-1} & \binom{n}{2}\lambda^{n-2} & \cdots & \binom{n}{k-1}\lambda^{n-k+1} \\ 0 & \lambda^n & n\lambda^{n-1} & \cdots & \binom{n}{k-2}\lambda^{n-k+2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & n\lambda^{n-1} \\ 0 & 0 & 0 & \cdots & \lambda^n \end{pmatrix}.$$

Furthermore, each eigenvalue λ of M appears in at least one of the Jordan blocks.

In case $L = \mathbb{Q}$, we may take K to be an algebraic number field. In particular, the eigenvalues of a rational matrix are algebraic. However, in this paper, the entries of our matrix are *algebraic functions*, and so too are the entries in Jordan normal form. We recall some basics of algebraic geometry and univariate algebraic functions required for the analysis in the single-parameter setting, and refer the reader to [5, 18] for further information.

Let $U \subseteq \mathbb{C}$ be a connected open set and $f : U \to \mathbb{C}$ a meromorphic function. We say that f is *algebraic over* $\mathbb{Q}(x)$ if there is a polynomial $P(x, y) \in \mathbb{Q}[x, y]$ such that $P(x, f(x)) = 0$ for all $x \in U$ where f is defined. Notice that a univariate algebraic function has finitely many zeros and poles, and furthermore, these zeros and poles (or zeros at ∞) are algebraic. Indeed, let $P(x, y) = a_d(x)y^d + \cdots + a_1(x)y + a_0(x)$, with $a_i \in \mathbb{Q}[x]$, be irreducible. Assuming that f vanishes at s, we have that $a_0(s) = 0$. There are only finitely many s for which this can occur. Furthermore, the function $1/f$ is meromorphic (on a possibly different domain U) and satisfies $y^d P(x, 1/y) = a_d(x) + \ldots + a_1(x)y^{d-1} + a_0(x)y^d$. We conclude that a pole of f (a zero of $1/f$) is a zero of $a_d(x)$.

Let $P(x, y) = \sum_{i=0}^{d} a_i(x)y^i \in \mathbb{Q}(x)[y]$. We say that $c \in \mathbb{C}$ is a *critical point* of P if either $a_d(c) = 0$ or the resultant $\operatorname{Res}_y(P, \frac{\partial P}{\partial y})$ vanishes at c. If P is irreducible, then it has only finitely many critical points since the resultant is a univariate non-zero polynomial.

Let M be a $(d \times d)$-matrix with univariate real algebraic functions as entries. Let its characteristic polynomial be $P(x, y) := \det(Iy - M)$ and write $c_1, \ldots, c_m \in \mathbb{C}$ for the critical points of the irreducible factors of P. Then there exist a connected open subset $U \subseteq \mathbb{C}$ such that $\mathbb{R} \setminus \{c_1, \ldots, c_m\} \subseteq U$, and d holomorphic functions $\lambda_1, \ldots, \lambda_d : U \to \mathbb{C}$ (not necessarily distinct) such that the characteristic polynomial P of M factors as

$$P(x, y) = (y - \lambda_1(x))(y - \lambda_2(x)) \cdots (y - \lambda_d(x))$$

for all points $x \in U$ (see, e.g., [21, Chapt. 1, Thm. 8.9]).

Let us fix a $(d \times d)$-matrix M and vectors u, v with univariate real algebraic entries. We thus have $M \in L^{d\times d}$, $u, v \in L^d$, for some finite field extension L of $\mathbb{Q}(x)$. Let $\mathbb{K}$ be fixed to an algebraic extension of L such that the characteristic polynomial of M splits into linear factors over the field $\mathbb{K}$. Then, over the field $\mathbb{K}$ we have the factorisation $M = C^{-1}JC$ with J in Jordan form. The eigenvalues of M, denoted $\lambda_1, \ldots, \lambda_k$, appear in the diagonal of J. Let the set of exceptional points, denoted $\mathcal{E}$, consist of the finite set $\{c_1, \ldots, c_m\}$, the poles of the entries of M, C, C^{-1}, J, u and v, and points where $\det C(s) = 0$ (i.e., $C(s)$ is singular).

Consider now a non-constant univariate algebraic function λ not necessarily real. In our analysis, we shall need to bound the height $h(\lambda(s))$ in terms of $h(s)$, as long as s is not a zero or a pole of λ. The following lemma shows $h(\lambda(s)) = \Theta(h(s))$:

▶ **Lemma 5.** *Let λ be a non constant algebraic function in $\mathbb{K}$. Then there exist effective constants $c_1, c_2, c_3, c_4 > 0$ such that for algebraic s not a zero or pole of λ we have*
$c_1 h(s) - c_2 \le h(\lambda(s)) \le c_3 h(s) + c_4$.

2.2.1 Multiplicative relations

Let $Y = \{\lambda_1, \ldots, \lambda_t\} \subset \mathbb{K}$ be a set of univariate algebraic functions.

▶ **Definition 6.** *A tuple $(a_1, \ldots, a_t) \in \mathbb{Z}^t$ for which $\lambda_1^{a_1} \cdots \lambda_t^{a_t} = 1$ identically, is called a* multiplicative relation. *A set of multiplicative relations is called* independent *if it is $\mathbb{Z}$-linearly independent as a subset of $\mathbb{Z}^t$. The set Y is said to be* multiplicatively dependent *if it satisfies a non-zero multiplicative relation. Otherwise Y is* multiplicatively independent. *The* rank *of Y, denoted* $\operatorname{rank} Y$, *is the size of the largest multiplicatively independent subset of Y.*

A tuple $(a_1, \ldots, a_t) \in \mathbb{Z}^t$, for which there exists $c \in \overline{\mathbb{Q}}$ such that $\lambda_1^{a_1} \cdots \lambda_t^{a_t} = c$ identically, is called a multiplicative relation modulo constants. *We say that Y is* multiplicatively dependent modulo constants *if it satisfies a non-zero multiplicative relation modulo constants. Otherwise Y is* multiplicatively independent modulo constants.

In particular, if $\operatorname{rank}\langle\lambda_1, \ldots, \lambda_t\rangle = 1$, then for each pair λ_i, λ_j, we have $\lambda_i^b = \lambda_j^a$ for some integers a, b not both zero. In the analysis that follows, we only need to distinguish between this case and $\operatorname{rank}\langle\lambda_1, \ldots, \lambda_t\rangle \ge 2$. We will also need to find multiplicative relations modulo constants between algebraic functions. These can be algorithmically determined and constructed as a consequence of the following proposition. To this end, let L and $L' \subseteq \mathbb{Z}^t$ be the set of multiplicative relations and multiplicative relations modulo constants on Y, respectively. Both L and L' are finitely generated as subgroups of $\mathbb{Z}^t$ under vector addition.

▶ **Proposition 7.** *Given a set $Y = \{\lambda_1, \ldots, \lambda_t\}$ of univariate algebraic functions, one can compute a generating set for both L and L'.*

Proof. This is essentially a special case of a result from [20]. Indeed, in Sect. 3.2, they show how to find the generators of the group L in case the λ_i are elements of a finitely generated field over $\mathbb{Q}$. We apply the result to the field $\mathbb{Q}(x, \lambda_1, \ldots, \lambda_t)$ to obtain the claim for the set L. For L', Case 3 of [20, Sect. 3.2] computes a generating set as an intermediate step in the computation of a basis of L. Specifically, L and L' are the respective kernels of the maps φ and $\tilde{\varphi}$ in [20, Sect. 3.2]. We give an alternative proof sketch specialised to univariate functions in the full version. ◀

3 The Multi-Parameter Orbit Problem is Skolem-hard

The *Skolem Problem* asks, given a order-k linear recurrence sequence $(u_n)_n$, uniquely defined by a recurrence relation $u_n = a_1 u_{n-1} + \cdots + a_k u_{n-k}$ for fixed $a_1, \ldots, a_k$ and initial points $u_1, \ldots, u_k$, whether there exists an n such that $u_n = 0$. The problem is famously not known to be decidable for orders at least 5, and problems which the Skolem problem reduce to are said to be *Skolem-hard.* We will now reduce the Skolem at order 5 to the two-parameter parametric orbit problem.

It suffices to only consider the instances of Skolem Problem at order 5 of the form $u_n = a\lambda_1^n + \overline{a\lambda_1^n} + b\lambda_2^n + \overline{b\lambda_2^n} + c\rho^n = 0$ with $|\lambda_1| = |\lambda_2| \geq |\rho|$ and $a, b, \lambda_1, \lambda_2 \in \overline{\mathbb{Q}}$, $c, \rho \in \overline{\mathbb{Q}} \cap \mathbb{R}$, as the instances of the Skolem Problem at order 5 that are not of this form are known to be decidable [36]. We may assume that $c = \rho = 1$ by considering the sequence $(u_n/c\rho^n)$ if necessary. We can also rewrite $u_n = A\mathrm{Re}\lambda_1^n + B\mathrm{Im}\lambda_1^n + C\mathrm{Re}\lambda_2^n + D\mathrm{Im}\lambda_2^n + 1$ for $A, B, C, D \in \overline{\mathbb{Q}} \cap \mathbb{R}$.

Let $u_n = a\lambda_1^n + \overline{a\lambda_1^n} + b\lambda_2^n + \overline{b\lambda_2^n} + 1 = A\mathrm{Re}\lambda_1^n + B\mathrm{Im}\lambda_1^n + C\mathrm{Re}\lambda_2^n + D\mathrm{Im}\lambda_2^n + 1$ be a hard instance of the Skolem Problem. Let $M = \mathrm{diag}\left(\begin{bmatrix}\mathrm{Re}\lambda_1 & -\mathrm{Im}\lambda_1 \\ \mathrm{Im}\lambda_1 & \mathrm{Re}\lambda_1\end{bmatrix}, \begin{bmatrix}\mathrm{Re}\lambda_2 & -\mathrm{Im}\lambda_2 \\ \mathrm{Im}\lambda_2 & \mathrm{Re}\lambda_2\end{bmatrix}\right)$, that is, the Real Jordan Normal Form of $\mathrm{diag}(\lambda_1, \overline{\lambda_1}, \lambda_2, \overline{\lambda_2})$. We set the starting point to be $u = [1\ 1\ 1\ 1]^\top$ and show how to define parametrized target vectors $v_1(s,t), \ldots, v_k(s,t)$ such that for all n, $u_n = 0$ if and only if there exist $s, t \in \mathbb{R}$ such that $M^n u = v_i(s,t)$ for some i. The Skolem Problem at order 5 then reduces to k instances of the two-parameter orbit problem.

The idea of our reduction is to first construct a semiagebraic set $Z \subseteq \mathbb{R}^4$, $Z = \bigcup_{i=1}^k Z_i$ such that $u_n = 0$ if and only if $(\mathrm{Re}\lambda_1^n, \mathrm{Im}\lambda_1^n, \mathrm{Re}\lambda_2^n, \mathrm{Im}\lambda_2^n) \in Z$, and each Z_i is a semialgebraic subset of $\mathbb{R}^4$ that can be described using two parameters and algebraic functions in two variables. Observing that $M^n s = (Re\lambda_1^n - \mathrm{Im}\lambda_1^n, \mathrm{Im}\lambda_1^n + \mathrm{Re}\lambda_1^n, Re\lambda_2^n - \mathrm{Im}\lambda_2^n, \mathrm{Im}\lambda_2^n + \mathrm{Re}\lambda_2^n)$, we then compute $v_i(s,t)$ from Z_i as follows. Suppose $Z_i = \{(x(s,t), y(s,t), z(s,t), u(s,t) : s, t. \in \mathbb{R}\}$. Then $v_i(s,t) = (x(s,t) - y(s,t), y(s,t) + x(s,t), u(s,t) - v(s,t), v(s,t) + u(s,t))$.

To compute Z, first observe that $\mathrm{Im}\lambda_2^n = \pm\sqrt{(\mathrm{Re}\lambda_1^n)^2 + (\mathrm{Im}\lambda_1^n)^2 - (\mathrm{Re}\lambda_2^n)^2}$ for all n as $|\lambda_1| = |\lambda_2|$. Motivated by this observation, let $S_+, S_- \subseteq \mathbb{R}^3$, $S_+ = \{(x,y,z) : Ax + By + Cz + D\sqrt{x^2 + y^2 - z^2} + 1 = 0\}$ and $S_- = \{(x,y,z) : Ax + By + Cz - D\sqrt{x^2 + y^2 - z^2} + 1 = 0\}$. We will choose $Z = \{(x, y, z, \sqrt{x^2 + y^2 - z^2}) : (x,y,z) \in S_+\} \cup \{(x, y, z, -\sqrt{x^2 + y^2 - z^2}) : (x,y,z) \in S_-\}$. It is easy to check that the above definition of Z satisfies the requirement that $u_n = 0$ if and only if $(\mathrm{Re}\lambda_1^n, \mathrm{Im}\lambda_1^n, \mathrm{Re}\lambda_2^n, \mathrm{Im}\lambda_2^n) \in Z$, and it remains to show that both S_+ and S_- can be parametrized using algebraic functions in two variables and two parameters. To this end, observe that S_+ and S_- are both semialgebraic subsets of $\mathbb{R}^3$, but are also contained in the algebraic set $S = \{(x,y,z) : (Ax + By + Cz + 1)^2 = D^2(x^2 + y^2 - z^2)\} \subseteq \mathbb{R}^3$. Since $S \neq \mathbb{R}^3$ (for example, $(0,0,0) \notin S$), and it is algebraic, S can have *dimension* (see [18] for a definition) at most 2. Hence S_+, S_- also have semialgebraic dimension at most 2. In the full version, we show that a semialgebraic subsets of $\mathbb{R}^3$ of dimension at most two can be written as a finite union of sets of the form $\{v(s,t) : s, t \in \mathbb{R}\}$, where v is an algebraic function. This completes the construction of Z and the description of the reduction.

4 Single Parameter Reachability: Overview of proof

In this section we show how to prove Theorem 2, that is, it is decidable, given a $(d \times d)$-matrix M, initial and target vectors u, v, whose entries are real algebraic functions all depending on a single parameter, whether there exist $s \in \mathbb{R}$ giving rise to a concrete matrix, initial and target $M(s) \in \mathbb{R}^{d\times d}, u(s), v(s) \in \mathbb{R}^d$, and a positive integer $n \in \mathbb{N}$, such that $M(s)^n u(s) = v(s)$.

In our case analysis, we often show that either there is a finite set of parameter values for which the constraints could hold, or place an upper bound on the n for which the constraints hold. The following proposition shows that the decidability of the problem in these cases is apparent:

▶ **Proposition 8.**

- *Given a finite set $S \subset \mathbb{R}$ it is decidable if there exists $(n, s) \in \mathbb{N} \times S$ s.t. $M(s)^n u(s) = v(s)$.*
- *Given $B \in \mathbb{N}$ it is decidable if there exists $n \leq B$ and $s \in \mathbb{R}$ s.t. $M(s)^n u(s) = v(s)$.*

Proof. The decidability of the first case is a consequence of the fact that a choice of parameter leads to a concrete matrix, thus giving an instance of the non-parametric Orbit Problem.

In the second case, for fixed n, one can observe that the matrix M^n is itself a matrix of real algebraic functions. Hence the equation $M^n u = v$ can be rewritten as equations $P_i(x) = 0$ for real algebraic P_i for $i = 1, \ldots, d$. For each equation the function is either identically zero, or vanishes at only finitely many s which can be determined, and one can check if there is an s in the intersection of the zero sets as i varies. Repeat for each $n \leq B$. ◀

As a consequence, for each n either $M^n u = v$ holds identically (for every s), or there are at most finitely many s such that $M(s)^n u(s) = v(s)$, and all such points are algebraic, as they must be the roots of the algebraic functions P_i.

Our approach will be to place the problem into Jordan normal form (Section 4.1), where we will observe that the problem can be handled if the resulting form is not diagonal (Section 4.2). Here the relation between the Weil height of an algebraic number and its image under an algebraic function are exploited to bound n (reducing to the second case of the proceeding proposition).

In the diagonal case the problem can be reformulated for algebraic functions λ_i, γ_i for $i = 1 \ldots, t$, whether there exist $(n, s) \in \mathbb{N} \times \mathbb{R} \setminus \mathcal{E}$ such that $\lambda_i^n(s) = \gamma_i(s)$ for all $i = 1, \ldots, t$, where $\mathcal{E}$ is a finite set of exceptional points. These exceptional points can be handled separately using the first case of the proceeding proposition.

To show decidability we will distinguish between the case where $\operatorname{rank}\langle \lambda_1, \ldots, \lambda_t \rangle$ is 1 and when it is greater than 2 (recall Definition 6). As discussed in the introduction, the most intriguing part of our development will be in the case of $\operatorname{rank}\langle \lambda_1, \ldots, \lambda_t \rangle \geq 2$, captured in the following lemma:

▶ **Lemma 9.** *Let $\lambda_1, \ldots, \lambda_t$ be algebraic functions in $\mathbb{K}$ and $\operatorname{rank}\langle \lambda_1, \ldots, \lambda_t \rangle \geq 2$. Given algebraic functions $\gamma_1, \ldots, \gamma_t$ in $\mathbb{K}$, then it is decidable whether there exist $(n, s) \in \mathbb{N} \times \mathbb{R} \setminus \mathcal{E}$ such that*

$$\lambda_i(s)^n = \gamma_i(s) \qquad \text{for all} \qquad i = 1, \ldots, t. \tag{2}$$

The proof of this lemma is shown in Section 5. Here we apply two specialised arguments, in the case of non-constant λ's we exploit the results of Bombieri, Masser, and Zannier [8, 9] to show there is a finite effective set of parameter values. In the case of constant λ's we reduce to an instance of Skolem's problem that we show is decidable, effectively bounding n.

It will then remain to prove a similar lemma for the case where the rank is 1. Here we will exploit the initial use of real algebraic functions, to ensure the presence of complex conjugates.

▶ **Lemma 10.** *Let $\lambda_1, \ldots, \lambda_t$ be algebraic functions in $\mathbb{K}$ and $\operatorname{rank}\langle\lambda_1, \ldots, \lambda_t\rangle = 1$. We assume that, if λ_i is complex then $\overline{\lambda_i}$ (the complex conjugate) also appears. Given algebraic functions $\gamma_1, \ldots, \gamma_t$ in $\mathbb{K}$, then it is decidable whether there exist $(n, s) \in \mathbb{N} \times \mathbb{R} \setminus \mathcal{E}$ such that $\lambda_i^n(s) = \gamma_i(s)$ for all $i = 1, \ldots, t$.*

The proof of this lemma (in Section 6), reduces the problem to a single equation ($t = 1$), for which we provide a specialised analysis on the behaviour of such functions that enable us to decide the existence of a solution.

In the remainder of this section we will show how to place the problem in the form of these two lemmas: first placing the matrix into Jordan normal form, eliminating the cases where the Jordan form is not diagonal and provide some simplifying assumptions for the proofs of Lemmas 9 and 10.

4.1 The parametric Jordan normal form

For every $s \in \mathbb{R}\setminus\mathcal{E}$ we have $M(s) = C^{-1}(s)J(s)C(s)$ and hence, for every $n \in \mathbb{N}$, $M^n(s)u(s) = v(s)$ if and only if $J^n(s)C(s)u(s) = C(s)v(s)$. On the other hand, deciding whether there exists $s \in \mathcal{E}$ with $M^n(s)u(s) = v(s)$ reduces to finitely many instances of the Kannan-Lipton Orbit Problem, which can be decided separately. We have thus reduced the parametric point-to-point reachability problem to the following one in case of a single parameter:

▶ **Problem 11.** Given a matrix $J \in \mathbb{K}^{d\times d}$ in Jordan normal form, and vectors $\tilde{u}$, $\tilde{v} \in \mathbb{K}^d$, decide whether there exists $(n, s) \in \mathbb{N} \times \mathbb{R} \setminus \mathcal{E}$ such that $J^n(s)\tilde{u}(s) = \tilde{v}(s)$.

▶ **Example 12.** Define $M = \begin{pmatrix} x+\frac{1}{2} & 0 & 0 \\ \frac{1}{2}-x & 1-x & 0 \\ 0 & x & 1 \end{pmatrix} \in \mathbb{Q}(x)^{3\times 3}$. Then the characteristic polynomial of M is $\det(yI - M) = (y - 1/2 - x)(y - 1)(y + x - 1)$. The irreducible factors have no critical points. Now over $\mathbb{K}$ we may write $M = C^{-1}JC$, where $J = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1-x & 0 \\ 0 & 0 & x+\frac{1}{2} \end{pmatrix}$, $C = \begin{pmatrix} 1 & 1 & 1 \\ \frac{1-2x}{4x-1} & -1 & 0 \\ \frac{2x}{1-4x} & 0 & 0 \end{pmatrix}$, and $C^{-1} = \begin{pmatrix} 0 & 0 & \frac{1}{2x}-2 \\ 0 & -1 & 1-\frac{1}{2x} \\ 1 & 1 & 1 \end{pmatrix}$. Notice that J is defined for all x, while C is not defined at $1/4$, and C^{-1} is not defined at 0 (notice also that $C(0)$ is not invertible). Therefore $\mathcal{E} = \{0, 1/4\}$. For $s \in \mathbb{R}\setminus\mathcal{E}$, all three are defined and we have $M(s) = C^{-1}(s)J(s)C(s)$, with $J(s)$ in Jordan normal form and $C(s)$ invertible.

Notice, for $1/4 \in \mathcal{E}$, we have $M(1/4) = R^{-1}KR$, where $K = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{3}{4} & 1 \\ 0 & 0 & \frac{3}{4} \end{pmatrix}$ and $R = \begin{pmatrix} 1 & 1 & 1 \\ -1 & -1 & 0 \\ -\frac{1}{4} & 0 & 0 \end{pmatrix}$. Notice here that $M(1/4)$ is non-diagonalisable (over $\overline{\mathbb{Q}}$), though M is (over $\mathbb{K}$).

Let $u = (u_1, u_2, u_3) \in \mathbb{Q}(x)^3$ and $v = (v_1, v_2, v_3) \in \mathbb{Q}(x)^3$. The problem of whether there exists $(n, s) \in \mathbb{N} \times \mathbb{R}$ for which $M(s)^n u(s) = v(s)$ is reduced to checking the problem at $s \in \mathcal{E}$, and to the associated problem $J^n(s)\tilde{u}(s) = \tilde{v}(s)$, where $\tilde{u} = \begin{pmatrix} u_1+u_2+u_3 \\ \frac{1-2x}{4x-1}u_1-u_2 \\ \frac{2x}{1-4x}u_1 \end{pmatrix}$, $\tilde{v} = \begin{pmatrix} v_1+v_2+v_3 \\ \frac{1-2x}{4x-1}v_1-v_2 \\ \frac{2x}{1-4x}v_1 \end{pmatrix}$, and $J^n = \begin{pmatrix} 1 & 0 & 0 \\ 0 & (1-x)^n & 0 \\ 0 & 0 & (x+\frac{1}{2})^n \end{pmatrix}$.

Let us establish some notation: assume $J = \langle J_1, \ldots, J_N\rangle$, corresponding to eigenvalues $\lambda_1, \ldots, \lambda_N$. Assume the dimension of Jordan block J_i is d_i, and let $\tilde{u}_{i,1}, \ldots, \tilde{u}_{i,d_i}$ be the coordinates of $\tilde{u}$ associated with the Jordan block J_i, where index 1 corresponds to the bottom of the block. Similarly, let $\tilde{v}_{i,1}, \ldots, \tilde{v}_{i,d_i}$ be the corresponding entries of the target.

Let us define the functions $\gamma_1, \dots, \gamma_N$ used in our reduction to Lemma 9 and Lemma 10. We let $\gamma_i(s) = \tilde{v}_{i,1}(s)/\tilde{u}_{i,1}(s)$, for $\tilde{u}_{i,1}(s) \neq 0$. If $\tilde{u}_{i,1}$ is not constant zero, then there are finitely many s where $\tilde{u}_{i,1}(s) = 0$, each of which can be handled explicitly. If some $\tilde{u}_{i,1}$ is the constant zero function, then there are two cases. Firstly, if $\tilde{v}_{i,1}$ is also the constant zero then we are in the degenerate case $\lambda_i^n \cdot 0 = 0$, and the row can be ignored. Secondly if $\tilde{v}_{i,1}$ is not constant zero, then there are only a finite number of s s.t. $0 = \tilde{v}_{i,1}(s)$. Each of these can be checked explicitly.

We say that an eigenvalue $\lambda \in \mathbb{K}$ (possibly constant) is a *generalised root of unity* if there exists an $a \in \mathbb{N}_{\geq 1}$, such that $\lambda^a(x)$ is a real-valued and non-negative function. Let $\operatorname{order}(\lambda)$ of a generalised root of unity λ be the minimal such a. Notice that any real function is a generalised root of unity with order at most 2. When we say an eigenvalue *is* a root of unity, then the eigenvalue is necessarily a constant function.

▶ **Lemma 13.** *To decide Problem 11 it suffices to assume that no λ_i is identically zero and that any λ_i which is a generalised root of unity is real and non-negative (in particular, the only roots of unity are exactly 1).*

Proof. If $\lambda_i = 0$, then $J_i^{d_i+n} = 0$ for all $n \in \mathbb{N}$, hence we only need to check $n \leq d_i$ and the s such that $\tilde{v}_{i,1}(s) = \cdots = \tilde{v}_{i,d_i}(s) = 0$ (unless this holds identically, in which case the constraints from this Jordan block can be removed).

Take $L = \operatorname{lcm}\{\operatorname{order}(\lambda_i) \mid \lambda_i \text{ is generalised root of unity}\}$. Then the reachability problem reduces to L problems: $(J^L)^n(J^k\tilde{u}(x)) = \tilde{v}(x)$ for every $k \in \{0, \dots, L-1\}$. The eigenvalue λ_i^L corresponding to $(J_i)^L$ is now real and non-negative if it is a generalised root of unity. ◀

4.2 Jordan cells of dimension larger than 1

First, we show decidability of the problem when some Jordan block has dimension at least 2:

▶ **Proposition 14.** *If there exists J_i such that $d_i > 1$, then Problem 11 is decidable.*

There are three cases not covered by the previous section: λ_i is not constant, λ_i is constant but not a root of unity, and $\lambda_i = 1$.

Let us start with the case where $\lambda_i \neq 1$, that is λ_i is a constant but not 1, or λ_i is not a constant. Here we can use the bottom two rows from the block to obtain:

$$\lambda_i^n(x)\tilde{u}_{i,1}(x) = \tilde{v}_{i,1}(x) \quad \text{and} \quad \lambda_i^n(x)\tilde{u}_{i,2}(x) + n\lambda_i^{n-1}(x)\tilde{u}_{i,1}(x) = \tilde{v}_{i,2}(x),$$

We reformulate these equations, defining algebraic function θ:

$$\lambda_i^n(x) = \gamma_i(x) = \tilde{v}_{i,1}(x)/\tilde{u}_{i,1}(x) \quad \text{and} \quad n = \theta(x) = \lambda_i(x)(\tilde{v}_{i,2}(x)/\tilde{v}_{i,1}(x) - \tilde{u}_{i,2}(x)/\tilde{u}_{i,1}(x))$$

Any roots or poles of $\tilde{u}_{i,1}, \tilde{u}_{i,2}, \tilde{v}_{i,1}, \tilde{v}_{i,2}, \lambda_i$ can be handled manually (and we already ensured $\tilde{u}_{i,1}$ is not identically zero). We can then apply the following lemma.

▶ **Lemma 15.** *Given algebraic functions λ, γ, θ in parameter x, with λ not a root of unity, then there is a bound on $n \in \mathbb{N}$ such that there exists an $s \in \overline{\mathbb{Q}}$ with $n = \theta(s)$ and $\lambda^n(s) = \gamma(s)$.*

Proof sketch. We sketch the case where λ is not a constant function, a similar (but distinct) approach is used for λ constant. Taking heights on $\lambda^n(s) = \gamma(s)$ we obtain $nh(\lambda(s)) = h(\gamma(s))$, applying Lemma 5 twice (on both λ and γ) we obtain $nh(s) = \Theta(h(s))$. In particular if n is large (say $n > A$) then $h(s)$ is bounded (say $h(s) < B$). Taking heights on $n = \theta(s)$ we obtain $\log(n) = h(n) = h(\theta(s)) = \Theta(h(s))$. If $n > A$ then $\log(n) \leq BC$. Hence $n \leq \max\{A, exp(BC)\}$. ◀

The remaining case where $\lambda_i = 1$ results only in an equation of the form $n = \theta(s)$, so $\lambda_j^n(s) = \gamma_j(s)$ can be taken from any other Jordan block where $\lambda_j \neq 1$ and again we apply Lemma 15 to place a bound on n.

4.3 Further simplifying assumptions for diagonal matrices

Henceforth, we may assume that J is a diagonal matrix resulting in the formulation of Lemmas 9 and 10: given eigenvalues $\lambda_1, \ldots, \lambda_t$ and so we want to know if there exists $(n, s) \in \mathbb{N} \times \mathbb{R} \setminus \mathcal{E}$ such that

$$\lambda_i^n(s) = \gamma_i(s) \qquad \text{for all} \qquad i = 1, \ldots, t \tag{3}$$

Finally we make some simplifications in Lemma 16:

▶ **Lemma 16.** *To decide Problem 11, it suffices to decide the problem with instances where the eigenvalues λ_i are distinct, that none of the λ_i's are identically zero, that none of the constant λ_i's are roots of unity, and every constant λ_i is associated with non-constant γ_i.*

Proof. Consider first the case that $\lambda_1 = \lambda_2$. If also $\gamma_1 = \gamma_2$ then the equations $\lambda_1^n = \gamma_1$ and $\lambda_2^n = \gamma_2$ are equivalent and one of them can be removed. Otherwise, if $\gamma_1 \neq \gamma_2$, the equations $\lambda_1^n = \gamma_1$ and $\lambda_2^n = \gamma_2$ can only have a common solution for $s \in \mathbb{R}$ with $\gamma_1(s) = \gamma_2(s)$, i.e., we can restrict to a finite set of parameters, in which case the problem becomes decidable.

We have already established, in Lemma 13, that none of the λ_i's are identically zero, and that the only constant root of unity is 1. Indeed if $\lambda_j = 1$ then we have $1^n = \gamma_j(s)$, which holds either at finitely many s or γ_j is the constant 1 and the constraint can be dropped.

If there exists i with constant λ_i (not a root of unity) and constant γ_i then there is at most a single n such that $\lambda_i^n = \gamma_i$. This n can be found using the Kannan-Lipton problem on the single constraint. The remaining constraints can be verified for this n using Proposition 8 to determine if they are simultaneously satisfiable. ◀

4.4 Multiplicative dependencies

To handle cases when the eigenvalues λ_i's are multiplicatively dependent, we often argue as in the following manner. Say $\lambda_1^{a_1} = \lambda_2^{a_2} \cdots \lambda_t^{a_t}$ with $a_1 \neq 0$. Consider the system

$$\lambda_i^{a_i}(s)^n = \gamma_i^{a_i}(s) \qquad \text{for all} \qquad i = 1, \ldots, t. \tag{4}$$

It is clear that the set E of solutions (n, s) to (3) is a subset of the set E' of solutions to (4). Furthermore, for $(n, s) \in E'$ we have $\gamma_1^{a_1}(s) = \lambda_1^{a_1 n}(s) = (\lambda_2^{a_2} \cdots \lambda_t^{a_t})^n(s) = \gamma_2^{a_2} \cdots \gamma_t^{a_t}(s)$.

We conclude that if $\gamma_1^{a_1} \neq \gamma_2^{a_2} \cdots \gamma_t^{a_t}$, then there can only be finitely many s solving (4), and thus the original problem, and so the problem becomes decidable. In case $\gamma_1^{a_1} = \gamma_2^{a_2} \cdots \gamma_t^{a_t}$, the first equation in (4) is redundant, and we may remove it. By repeating the process we obtain a system of the form (4) where the λ_i are multiplicatively independent, and the solutions to it contain all the solutions to the original system.

Now we face the problem of separating solutions to (3) from the solutions to (4). If either of the sets $\{n \colon (n, s) \in E'\}$ or $\{s \colon (n, s) \in E'\}$ is finite and effectively enumerable, we can clearly decide whether E is empty or not, utilising either Kannan–Lipton or Proposition 8 finitely many times. This happens in the majority of cases. In the case that both the above sets are unbounded, we bound the suitable n in case $\operatorname{rank}\{\lambda_1, \ldots, \lambda_t\} \geq 2$ in Section 5. For the case of $\operatorname{rank}\{\lambda_1, \ldots, \lambda_t\} \leq 1$ we give a separate argument in Section 6.

5 The case of $\operatorname{rank}\langle \lambda_1, \ldots, \lambda_t \rangle \geq 2$

In this section we recall and prove the following Lemma 9:

▶ **Lemma 9.** *Let $\lambda_1, \ldots, \lambda_t$ be algebraic functions in $\mathbb{K}$ and $\operatorname{rank}\langle \lambda_1, \ldots, \lambda_t \rangle \geq 2$. Given algebraic functions $\gamma_1, \ldots, \gamma_t$ in $\mathbb{K}$, then it is decidable whether there exist $(n, s) \in \mathbb{N} \times \mathbb{R} \setminus \mathcal{E}$ such that*

$$\lambda_i(s)^n = \gamma_i(s) \qquad \text{for all} \qquad i = 1, \ldots, t. \tag{2}$$

By Lemma 16 we may assume that none of λ_i's are identically zero or a root of unity.

5.1 All λ_i's constant

In this section we sketch the proof for the case where λ_i's are all constant. We reduce to a special case of the Skolem problem, but show that this particular instance is decidable. Since rank ≥ 2, we have at least two constraints and so there are constants λ_1 and λ_2, not roots of unity, and multiplicatively independent, with γ_1, γ_2 not constant.

▶ **Lemma 17.** *Suppose λ_1, λ_2 are constant, not roots of unity, multiplicatively independent, and that γ_1, γ_2 are non-constant functions. Then the system $\lambda_1^n = \gamma_1(s)$, $\lambda_2^n = \gamma_2(s)$ has only finitely many solutions.*

Proof Sketch. Let the minimal polynomials over $\overline{\mathbb{Q}}[x, y]$ of γ_1 and γ_2 be P_1 and P_2 with $P_i \in \overline{\mathbb{Q}}[x, y_i]$. The polynomials P_1 and P_2 have no common factors as elements of $\overline{\mathbb{Q}}[x, y_1, y_2]$. Eliminating x from these polynomials we get a non-zero polynomial $P \in \overline{\mathbb{Q}}[y_1, y_2]$ for which $P(\alpha_1, \alpha_2) = 0$ for all $\alpha_1 = \gamma_1(s)$ and $\alpha_2 = \gamma_2(s)$, $s \in U$. The sequence $(u_n)_{n=0}^{\infty}$, with

$$u_n = P(\lambda_1^n, \lambda_2^n) = \sum_{k,\ell} a_{k,\ell} (\lambda_1^k \lambda_2^\ell)^n,$$

$a_{k,\ell} \in \overline{\mathbb{Q}}$, is a linear recurrence sequence over $\overline{\mathbb{Q}}$, and we wish to characterise those n for which $u_n = 0$. By the famous Skolem–Mahler–Lech theorem (see, e.g., [11]), the set of such n is the union of a finite set and finitely many arithmetic progressions. Furthermore, it is decidable whether such a sequence admits infinitely many elements, and all the arithmetic progressions can be effectively constructed [7]. But, in general, the elements of the finite set are not known to be effectively enumerable – solving the Skolem problem for arbitrary LRS essentially reduces to checking whether this finite set is empty. However, the case at hand can be handled using now standard techniques involving powerful results from transcendental number theory, such as Baker's theorem for linear forms in logarithms, and similar results on linear forms in p-adic logarithms (see, e.g., [34, 39]). We show there exists an effectively computable $n_0 \in \mathbb{N}$ such that $u_n \neq 0$ for all $n \geq n_0$. We give a brief sketch (a detailed proof appears in the full version):

Assuming first that $|\lambda_1|$ and $|\lambda_2|$ are multiplicatively independent, it is evident that the modulus of u_n grows as $c\alpha^n + o(\alpha^n)$ for some $c \in \mathbb{R}_+$, where α is the maximal modulus of the terms $\lambda_1^k \lambda_2^\ell$ (there is only one term with this modulus). One can straightforwardly compute an upper bound on any n for which $u_n = 0$.

If the values $|\lambda_1|$ and $|\lambda_2|$ are multiplicatively dependent but neither is of modulus 1, we may again use an asymptotic argument. For this, we need Baker's theorem on linear forms in logarithms to show that a (related) sequence grows in modulus as $c\alpha^n / n^D + o(\beta^n)$, with $\beta < \alpha$ and effectively computable constants c, D. On the other hand, if $|\lambda_i| = 1$ but λ_1 is an algebraic integer (a root of a monic polynomial with coefficients in $\mathbb{Z}$), then it will have a

Galois conjugate (roots of the minimal polynomial of λ_1) $\tilde{\lambda_1}$ with $|\tilde{\lambda_1}| > 1$. Hence a suitable Galois conjugate of the sequence (u_n) will be of the form considered in the previous case, and the zeros of (u_n) and $(\tilde{u}_n)$ coincide. The asymptotic argument can be applied to $(\tilde{u}_n)$.

The final case is when λ_1 and λ_2 are not algebraic integers. We turn to the theory of prime-ideal decompositions of the numbers λ and argue, employing a version of Baker's theorem for p-adic valuations (as in [39]) to conclude similarly that the n for which $u_n = 0$ are effectively bounded above. ◀

5.2 At least one non-constant

Henceforth, we can assume that at least one λ_i is non-constant. We may take the λ_i's to be multiplicatively independent with $t \geq 2$, otherwise consider a multiplicatively independent subset of the functions: it always has at least two elements by the assumption on rank, and, furthermore, at least one of them is not constant. The removal of equations will be done as described in Section 4.4; here we show that there are only finitely many n giving solutions (n, s) to the reduced system, so we need not worry about creating too many new solutions.

The following theorems are the main technical results from the literature utilised in the arguments that follow, formulated in a way to suit our needs. Here $\mathcal{C}(\overline{\mathbb{Q}})$ denotes the set of algebraic points in $\overline{\mathbb{Q}}^d$ on an algebraic set $\mathcal{C} \subseteq \mathbb{C}^d$.

▶ **Theorem 18** ([8, Theorem 2])**.** *Let $\mathcal{C}$ be an* absolutely irreducible *(irreducible in $\overline{\mathbb{Q}}(x)$) curve defined over $\overline{\mathbb{Q}}$ in $\mathbb{C}^d$. Assume that the coordinates of the curve are multiplicatively independent modulo constants (i.e., the points $(x_1, \ldots, x_d) \in \mathcal{C}(\overline{\mathbb{Q}})$ do not satisfy $x_1^{a_1} \cdots x_d^{a_d} = c$ identically for any $(a_1, \ldots, a_d) \in \mathbb{Z}^d \setminus \vec{0}$, $c \in \overline{\mathbb{Q}}$). Then the points $(x_1, \ldots, x_d) \in \mathcal{C}(\overline{\mathbb{Q}})$ for which $x_1, \ldots, x_d$ satisfy at least two independent multiplicative relations form a finite set.*

We note that given the curve $\mathcal{C}$, the finite set of points $(x_1, \ldots, x_d)$ on $\mathcal{C}$ for which $x_1, \ldots, x_d$, satisfy at least two independent multiplicative relations can be effectively constructed. Indeed, this is explicitly mentioned in the last paragraph of the introduction of [8]: the proof goes by showing effective bounds on the degree and height of such points.

Theorem 18 holds for curves in $\mathbb{C}^d$ for arbitrary d. If one allows the coordinates on the curve to satisfy a non-trivial multiplicative relation, then there can be infinitely many such points [8]. On the other hand, in [9] Bombieri, Masser, and Zannier consider relaxing the assumption of multiplicative independence modulo constants to multiplicative independence and conjecture that the conclusion of the above theorem still holds [9, Conj. A]. Supporting the conjecture, [9] proves a theorem which will suffice for us.

▶ **Theorem 19.** *Let $\mathcal{C}$ be an absolutely irreducible curve in $\mathbb{C}^d$ defined over $\overline{\mathbb{Q}}$. Assume that the the coordinates of the curve are multiplicatively independent, but $\mathcal{C}$ is contained in a set of the form $\vec{b}H$, where H is the set of points in $\overline{\mathbb{Q}}^d$ satisfying at least $d-3$ independent multiplicative relations*[2]*. Then the points $(x_1, \ldots, x_d) \in \mathcal{C}(\overline{\mathbb{Q}})$ for which $x_1, \ldots, x_d$ satisfy at least two independent multiplicative relations form a finite set.*

Again the finite set of points can be effectively computed.[3]

Let us proceed case by case.

[2] With $b = (b_1, \ldots, b_k)$, here $\vec{b}H = \{(b_1x_1, \ldots, b_dx_d) \colon (x_1, \ldots, x_k) \in H\}$ is a coset of a *subgroup of dimension at most* 3 in the terminology of [9].

[3] In [8, 9] the proof is given for $d \geq 4$, and is constructive, while the case of $d = 3$ is attributed to a (non-constructive) result of Liardet [31]. A completely effective proof of the case can be found in [6].

▶ **Lemma 20.** *Assume that $\{\lambda_1, \dots, \lambda_t\}$ is multiplicatively dependent modulo constants, but is multiplicatively independent. Then there exists a computable constant n_0 such that system* (2) *admits no solutions for $n > n_0$.*

We may now focus on sets $\{\lambda_1, \dots, \lambda_t\}$ that are multiplicatively independent modulo constants. We still might have multiplicative dependencies between the λ_i and γ_i. We take care of these cases in the remainder of this section.

▶ **Lemma 21.** *Assume that $\{\lambda_1, \lambda_2, \gamma_1, \gamma_2\}$ is multiplicatively independent. Then system* (2) *admits only finitely many solutions, all of which can be effectively enumerated.*

Proof. We show that the set of s for which the equality can hold is finite and such s can be computed. We employ the powerful Theorems 18 and 19 of Bombieri, Masser, and Zannier, from which the claim is immediate. We first prime the situation as follows.

Let that λ_1, λ_2, γ_1, γ_2 have minimal polynomials $P_1 \in \mathbb{Q}[x, x_1]$, $P_2 \in \mathbb{Q}[x, x_2]$, $P_3 \in \mathbb{Q}[x, x_3]$, $P_4 \in \mathbb{Q}[x, x_4]$, respectively. Eliminating x from P_1 and P_2 (resp., P_3, P_4), we get a polynomial $Q_1 \in \mathbb{Q}[x_1, x_2]$ (resp., $Q_2 \in \mathbb{Q}[x_1, x_3]$, $Q_3 \in \mathbb{Q}[x_1, x_4]$) for which we have $Q_1(\lambda_1(x), \lambda_2(x)) = 0$ (resp., $Q_2(\lambda_1(x), \gamma_1(x)) = 0$, $Q_3(\lambda_1(x), \gamma_2(x)) = 0$) for all x. Let $\mathcal{C}$ be the curve defined by $\mathcal{C} := \{(x_1, x_2, x_3, x_4) \in \mathbb{C}^4 \colon Q_1(x_1, x_2) = Q_2(x_1, x_3) = Q_3(x_1, x_4) = 0\}$ and consider any of its finitely many absolutely irreducible components $\mathcal{C}'$. We are now interested in the pairs of multiplicative relations $(n, 0, -1, 0)$ and $(0, n, 0, -1)$ (corresponding to $x_1^n = x_3$, $x_2^n = x_4$), for $n \geq 1$, along the curve $\mathcal{C}'$. Indeed, for any fixed n, the two relations are independent in $\overline{\mathbb{Q}}^4$, i.e., neither is a consequence of the other, as they involve disjoint sets of coordinates.

First assume that $\lambda_1, \lambda_2, \gamma_1, \gamma_2$ are multiplicatively independent modulo constants. Then so are the points on the curve $\mathcal{C}'$, and the result follows from Theorem 18 as the result is constructive.

Otherwise $\lambda_1, \lambda_2, \gamma_1, \gamma_2$ are multiplicatively dependent modulo constants but are multiplicatively independent. Then $\mathcal{C}'$ is contained in a set of the form $\vec{b}H$, where H satisfies at least one multiplicative relation. Applying Theorem 19 with $d = 4$, the points on $\mathcal{C}'$ satisfying $x_1^n = x_3$ and $x_2^n = x_4$ for any $n \geq 1$, form an effectively constructable finite set. ◀

To complete the proof of Lemma 9, we need to show the claim holds when $\lambda_1, \lambda_2, \gamma_1, \gamma_2$ are multiplicatively dependent, while λ_1 and λ_2 are multiplicatively independent modulo constants. The proof goes along the same lines as in the above with some extra technicalities.

▶ **Lemma 22.** *Assume that λ_1, λ_2, γ_1, γ_2 are multiplicatively dependent, while λ_1, λ_2 are multiplicatively independent modulo constants. Then there exists a computable constant n_0 such that system* (2) *admits no solutions for $n > n_0$.*

6 The case of $\operatorname{rank}\langle\lambda_1, \dots, \lambda_t\rangle = 1$

This section recalls and sketches the proof of Lemma 10.

▶ **Lemma 10.** *Let $\lambda_1, \dots, \lambda_t$ be algebraic functions in $\mathbb{K}$ and $\operatorname{rank}\langle\lambda_1, \dots, \lambda_t\rangle = 1$. We assume that, if λ_i is complex then $\overline{\lambda_i}$ (the complex conjugate) also appears. Given algebraic functions $\gamma_1, \dots, \gamma_t$ in $\mathbb{K}$, then it is decidable whether there exist $(n, s) \in \mathbb{N} \times \mathbb{R} \setminus \mathcal{E}$ such that $\lambda_i^n(s) = \gamma_i(s)$ for all $i = 1, \dots, t$.*

As sketched in Section 4.4, since there is a multiplicative dependence between functions, we first show that, without loss of generality, there is a single equation $\lambda^n(s) = \gamma(s)$.

▶ **Lemma 23.** *Suppose* $\operatorname{rank}\langle\lambda_1,\dots,\lambda_t\rangle = 1$*, then whether there is a solution* $(n, s) \in \mathbb{N}\times\mathbb{R}\setminus\mathcal{E}$ *to* $\lambda_i^n(s) = \gamma_i(s)$ *for all* $i = 1,\dots,t$ *reduces to instances with* $t = 1$*.*

We then separate into the case where λ is real and the case where λ is complex. Let us start by assuming λ is a real function.

▶ **Lemma 24.** *Given real algebraic functions* λ *and* γ*, it is decidable whether there exists* $(n, s) \in \mathbb{N} \times \mathbb{R} \setminus \mathcal{E}$ *such that* $\lambda^n(s) = \gamma(s)$*.*

Proof Sketch. The interesting case occurs on an interval $S = (s_0, s_1)$ on which $0 < \lambda(s), \gamma(s) < 1$ for $s \in S$. Other cases either reduce to this case, or occur for finitely many s which can be checked independently. The function $\gamma(s)$ is fixed between s_0, s_1. Each point $\lambda(s)^n$ decreases with every n. One can test for each n whether the lines $\lambda(s)$ and $\gamma(s)$ intersect, or one can find some bound n_0 after which $\lambda(s)^n < \gamma(s)$ for all $s \in S$ and $n > n_0$, so one can be sure there is no solution. ◀

Secondly, we consider the case λ takes on complex values. In this case, since λ_i was a complex eigenvalue of M, then so too is its conjugate $\overline{\lambda_i}$, yet λ_i and $\overline{\lambda_i}$ are multiplicatively dependent, in which case it turns out that $|\lambda| = 1$.

▶ **Lemma 25.** *Let* λ *and* γ *be algebraic functions. Assume* λ *is not real, non-zero, not a root of unity, and of modulus 1. The equation* $\lambda(s)^n = \gamma(s)$ *admits solutions as follows. If* γ *is not of modulus* 1 *constantly, then there are finitely many* s*. If* γ *is of modulus 1 identically and* λ *is constant, then there are infinitely many solutions and such a solution can be effectively found. Finally, if* λ *is not constant, then the equation admits a solution for all* $n \geq n_0$*, and* n_0 *is computable.*

Proof Sketch. The interesting case turns outs to be when λ and γ both define arcs on a unit circle. By taking powers of λ the arc grows, and eventually encompasses the arc defined by γ. The intermediate value theorem then implies there is an s satisfying $\lambda^n(s) = \gamma(s)$. ◀

References

1 S. Akshay, Timos Antonopoulos, Joël Ouaknine, and James Worrell. Reachability problems for Markov chains. *Inf. Process. Lett.*, 115(2):155–158, 2015. doi:10.1016/j.ipl.2014.08.013.

2 Shaull Almagor, Joël Ouaknine, and James Worrell. The Polytope-Collision Problem. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80 of *LIPIcs*, pages 24:1–24:14, Dagstuhl, Germany, 2017. doi:10.4230/LIPIcs.ICALP.2017.24.

3 Christel Baier, Florian Funke, Simon Jantsch, Toghrul Karimov, Engel Lefaucheux, Joël Ouaknine, Amaury Pouly, David Purser, and Markus A. Whiteland. Reachability in Dynamical Systems with Rounding. In Nitin Saxena and Sunil Simon, editors, *40th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2020)*, volume 182 of *LIPIcs*, pages 36:1–36:17, Dagstuhl, Germany, 2020. doi:10.4230/LIPIcs.FSTTCS.2020.36.

4 Ezio Bartocci, Radu Grosu, Panagiotis Katsaros, C. R. Ramakrishnan, and Scott A. Smolka. Model repair for probabilistic systems. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 326–340, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

5 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

6 Attila Bérczes, Kálmán Gyory, Jan-Hendrik Evertse, and Corentin Pontreau. Effective results for points on certain subvarieties of tori. *Mathematical Proceedings of the Cambridge Philosophical Society*, 147(1):69, 2009.

7 Jean Berstel and Maurice Mignotte. Deux propriétés décidables des suites récurrentes linéaires. *Bulletin de la Société Mathématique de France*, 104:175–184, 1976. doi:10.24033/bsmf.1823.

8 Enrico Bombieri, David Masser, and Umberto Zannier. Intersecting a curve with algebraic subgroups of multiplicative groups. *International Mathematics Research Notices*, 1999(20):1119–1140, January 1999. doi:10.1155/S1073792899000628.

9 Enrico Bombieri, David Masser, and Umberto Zannier. Intersecting curves and algebraic subgroups: conjectures and more results. *Transactions of the American Mathematical Society*, 358(5):2247–2257, 2006. doi:10.1090/S0002-9947-05-03810-9.

10 Jin-Yi Cai, Richard J Lipton, and Yechezkel Zalcstein. The complexity of the ABC problem. *SIAM Journal on Computing*, 29(6):1878–1888, 2000.

11 John W. S. Cassels. *Local Fields.* London Mathematical Society Student Texts. Cambridge University Press, 1986. doi:10.1017/CBO9781139171885.

12 Milan Češka, Frits Dannenberg, Marta Kwiatkowska, and Nicola Paoletti. Precise parameter synthesis for stochastic biochemical systems. In Pedro Mendes, Joseph O. Dada, and Kieran Smallbone, editors, *Computational Methods in Systems Biology*, pages 86–98, Cham, 2014. Springer International Publishing.

13 Rohit Chadha, Vijay Anand Korthikanti, Mahesh Viswanathan, Gul Agha, and YoungMin Kwon. Model checking MDPs with a unique compact invariant set of distributions. In *Eighth International Conference on Quantitative Evaluation of Systems, QEST 2011, Aachen, Germany, 5-8 September, 2011*, pages 121–130. IEEE Computer Society, 2011. doi:10.1109/QEST.2011.22.

14 Ventsislav Chonev, Joël Ouaknine, and James Worrell. The orbit problem in higher dimensions. In *Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '13, page 941–950, New York, NY, USA, 2013. doi:10.1145/2488608.2488728.

15 Ventsislav Chonev, Joël Ouaknine, and James Worrell. The polyhedron-hitting problem. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '15, page 940–956, USA, 2015. Society for Industrial and Applied Mathematics.

16 Ventsislav Chonev, Joël Ouaknine, and James Worrell. On the Skolem Problem for Continuous Linear Dynamical Systems. In Ioannis Chatzigiannakis, Michael Mitzenmacher, Yuval Rabani, and Davide Sangiorgi, editors, *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*, volume 55 of *LIPIcs*, pages 100:1–100:13, Dagstuhl, Germany, 2016. doi:10.4230/LIPIcs.ICALP.2016.100.

17 Henri Cohen. *A Course in Computational Algebraic Number Theory.* Springer Publishing Company, Incorporated, 2010.

18 David A. Cox, John Little, and Donal O'Shea. *Ideals, varieties, and algorithms - an introduction to computational algebraic geometry and commutative algebra.* Undergraduate texts in mathematics. Springer, 2 edition, 1997.

19 Murat Cubuktepe, Nils Jansen, Sebastian Junges, Joost-Pieter Katoen, and Ufuk Topcu. Synthesis in pMDPs: A tale of 1001 parameters. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis*, pages 160–176, Cham, 2018. Springer International Publishing.

20 Harm Derksen, Emmanuel Jeandel, and Pascal Koiran. Quantum automata and algebraic groups. *Journal of Symbolic Computation*, 39(3):357–371, 2005. Special issue on the occasion of MEGA 2003. doi:10.1016/j.jsc.2004.11.008.

21 Otto Foster. *Compact Riemann Surfaces*, volume 81 of *Graduate Textbooks in Mathematics.* Springer, 1981. doi:10.1007/978-1-4612-5961-9.

22 Robert Givan, Sonia Leach, and Thomas Dean. Bounded-parameter Markov decision processes. *Artificial Intelligence*, 122(1):71–109, 2000. doi:10.1016/S0004-3702(00)00047-3.

23 Michael A. Harrison. *Lectures on Linear Sequential Machines.* Academic Press, Inc., USA, 1969.

24 Bengt Jonsson and Kim Guldstrand Larsen. Specification and refinement of probabilistic processes. In *Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS '91), Amsterdam, The Netherlands, July 15-18, 1991*, pages 266–277. IEEE Computer Society, 1991. doi:10.1109/LICS.1991.151651.

25 Sebastian Junges, Erika Abraham, Christian Hensel, Nils Jansen, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. Parameter synthesis for Markov models, 2019.

26 Ravindran Kannan and Richard J. Lipton. Polynomial-time algorithm for the orbit problem. *J. ACM*, 33(4):808–821, 1986. doi:10.1145/6490.6496.

27 Vijay Anand Korthikanti, Mahesh Viswanathan, Gul Agha, and YoungMin Kwon. Reasoning about MDPs as transformers of probability distributions. In *QEST 2010, Seventh International Conference on the Quantitative Evaluation of Systems, Williamsburg, Virginia, USA, 15-18 September 2010*, pages 199–208. IEEE Computer Society, 2010. doi:10.1109/QEST.2010.35.

28 Igor Kozine and Lev Utkin. Interval-valued finite Markov chains. *Reliable Computing*, 8:97–113, April 2002. doi:10.1023/A:1014745904458.

29 YoungMin Kwon and Gul Agha. Linear inequality LTL (iLTL): A model checker for discrete time Markov chains. In Jim Davies, Wolfram Schulte, and Mike Barnett, editors, *Formal Methods and Software Engineering*, pages 194–208, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

30 Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. Parametric probabilistic transition systems for system design and analysis. *Formal Asp. Comput.*, 19:93–109, March 2007. doi:10.1007/s00165-006-0015-2.

31 Pierre Liardet. Sur une conjecture de Serge Lang. In *Journées arithmétiques de Bordeaux*, number 24–25 in Astérisque. Société mathématique de France, 1975. URL: www.numdam.org/item/AST_1975__24-25__187_0/.

32 Rupak Majumdar, Mahmoud Salamati, and Sadegh Soudjani. On Decidability of Time-Bounded Reachability in CTMDPs. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *LIPIcs*, pages 133:1–133:19, Dagstuhl, Germany, 2020. doi:10.4230/LIPIcs.ICALP.2020.133.

33 Maurice Mignotte. *Some Useful Bounds*, pages 259–263. Springer Vienna, Vienna, 1982. doi:10.1007/978-3-7091-3406-1_16.

34 Maurice Mignotte, Tarlok N. Shorey, and Robert Tijdeman. The distance between terms of an algebraic recurrence sequence. *Journal für die reine und angewandte Mathematik*, 1984(349):63–76, 01 May. 1984. doi:10.1515/crll.1984.349.63.

35 Alina Ostafe and Igor Shparlinski. On the Skolem problem and some related questions for parametric families of linear recurrence sequences, 2020.

36 Joël Ouaknine and James Worrell. Decision problems for linear recurrence sequences. In Alain Finkel, Jérôme Leroux, and Igor Potapov, editors, *Reachability Problems*, pages 21–28, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-33512-9_3.

37 Shashank Pathak, Erika Ábrahám, Nils Jansen, Armando Tacchella, and Joost-Pieter Katoen. A greedy approach for the efficient repair of stochastic models. In Klaus Havelund, Gerard Holzmann, and Rajeev Joshi, editors, *NASA Formal Methods*, pages 295–309, Cham, 2015. Springer International Publishing.

38 Jakob Piribauer and Christel Baier. On Skolem-Hardness and Saturation Points in Markov Decision Processes. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming (ICALP 2020)*, volume 168 of *LIPIcs*, pages 138:1–138:17, Dagstuhl, Germany, 2020. doi:10.4230/LIPIcs.ICALP.2020.138.

39 Nikolay K. Vereshchagin. Occurrence of zero in a linear recursive sequence. *Mathematical notes of the Academy of Sciences of the USSR*, 38:609–615, 1985.

40 Michel Waldschmidt. *Heights of Algebraic Numbers*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. doi:10.1007/978-3-662-11569-5_3.

Scope-Bounded Reachability in Valence Systems

Aneesh K. Shetty
Department of Computer Science & Engineering IIT Bombay, India

S. Krishna
Department of Computer Science & Engineering IIT Bombay, India

Georg Zetzsche
Max Planck Institute for Software Systems, Kaiserslautern, Germany

Abstract

Multi-pushdown systems are a standard model for concurrent recursive programs, but they have an undecidable reachability problem. Therefore, there have been several proposals to underapproximate their sets of runs so that reachability in this underapproximation becomes decidable. One such underapproximation that covers a relatively high portion of runs is *scope boundedness*. In such a run, after each push to stack i, the corresponding pop operation must come within a bounded number of visits to stack i.

In this work, we generalize this approach to a large class of infinite-state systems. For this, we consider the model of valence systems, which consist of a finite-state control and an infinite-state storage mechanism that is specified by a finite undirected graph. This framework captures pushdowns, vector addition systems, integer vector addition systems, and combinations thereof. For this framework, we propose a notion of scope boundedness that coincides with the classical notion when the storage mechanism happens to be a multi-pushdown.

We show that with this notion, reachability can be decided in PSPACE for every storage mechanism in the framework. Moreover, we describe the full complexity landscape of this problem across all storage mechanisms, both in the case of (i) the scope bound being given as input and (ii) for fixed scope bounds. Finally, we provide an almost complete description of the complexity landscape if even a description of the storage mechanism is part of the input.

2012 ACM Subject Classification Theory of computation → Models of computation; Theory of computation → Formal languages and automata theory

Keywords and phrases multi-pushdown systems, underapproximations, valence systems, reachability

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.29

Related Version *Full version with proofs*: https://arxiv.org/abs/2108.00963

1 Introduction

Multi-pushdown systems are a natural model for recursive programs with threads that communicate via shared memory. Unfortunately, even safety verification (state reachability) is undecidable for this model [21]. However, by considering *underapproximations* of the set of all executions, it is still possible to discover safety violations. The first such underapproximation in the literature was *bounded context switching* [20]. Here, one only considers executions that switch between threads a bounded number of times. In terms of multi-pushdown systems, this places a bound on the number of times we can switch between stacks.

One underapproximation that covers a relatively large portion of all executions and still permits decidable reachability is *scope-boundedness* as proposed by La Torre, Napoli, and Parlato [23, 25]. Here, instead of bounding the number of context switches across the entire run, we bound the number of context switches *per letter on a stack* (i.e. procedure execution). More precisely, whenever we push a letter on some stack i, then we can switch back to stack i at most k times before we have to pop that letter again. This higher coverage of runs comes at the cost of higher complexity: While reachability with bounded context switching is NP-complete [12, 20], the scope-bounded reachability problem is PSPACE-complete (if the number of pushdowns or the scope bound is part of the input) [25].

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 29; pp. 29:1–29:19
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Aside from multi-pushdown systems, there is a wide variety of other infinite-state models that are used to model program behaviours. For these, reachability problems are also sometimes undecidable or have prohibitively high complexity. For example, vector addition systems with states (VASS) is one of the most prominent models for concurrent systems, but its reachability problem has non-elementary complexity [8]. This raises the question of whether underapproximations for multi-pushdown systems can be interpreted in other infinite-state systems and what complexity would ensue.

The notion of bounded context switching has recently been generalized to a large class of infinite-state systems [19], in the framework of *valence systems over graph monoids*. These consist of a finite-state control that has access to a storage mechanism. The shape of this storage mechanism is described by a finite, undirected graph. By choosing an appropriate graph, one can realize many infinite-state models. Examples include (multi-)pushdown systems, VASS, integer VASS, but also combinations thereof, such as pushdown VASS [17] and sequential recursive Petri nets [16]. Under this notion, bounded context reachability is in NP for each graph, and thus each storage mechanism in the framework [19]. Moreover, the paper [19] presents some subclasses of graphs for which bounded context reachability has lower complexity (NL or P). However, the exact complexity of reachability under bounded context switching remains open in many cases, such as the path with four nodes [19].

Contribution. We present an *abstract notion of scope-bounded runs* for valence systems over graph monoids. As we show, this notion always leads to a reachability problem decidable in PSPACE. In particular, our notion applies to all infinite-state models mentioned above. Moreover, applied to multi-pushdown systems, it coincides with the notion of La Torre, Napoli, and Parlato.

We also obtain an almost complete complexity landscape of scope-bounded reachability. First, we show that if both (i) the graph Γ describing the storage mechanism and (ii) the scope bound k are part of the input, the problem is PSPACE-complete. Second, we study how the complexity depends on the employed storage mechanism. We show that for each Γ, the problem is either NL-complete, P-complete, or PSPACE-complete, depending on Γ (Corollary 4.2). Since the complexity drops below PSPACE only in extremely restricted cases, we also study the setting where the scope bound k is fixed. In this case, we show that the problem is either NL-complete or P-complete, depending on Γ (Corollary 4.4).

Finally, applying scope-boundedness to classes of infinite-state systems requires understanding the complexity if Γ is drawn from an infinite class of graphs. For example, for each fixed dimension d, there is a graph Γ_d such that valence automata over Γ_d correspond to VASS of dimension d. The class of *all* VASS (of arbitrary dimension), however, corresponds to valence automata over all cliques. Thus, we also study scope-bounded reachability if Γ is restricted to a class of graphs $\mathcal{G}$. Under a mild assumption on $\mathcal{G}$, we again obtain a complexity trichotomy of NL-, P-, or PSPACE-completeness, both for k as input (Theorem 4.1) and for fixed k (Theorem 4.3). In fact, all results mentioned above follow from these general results.

Related work. Similar in spirit to our work are the lines of research on systems with bounded tree-width by Madhusudan and Parlato [18] and on bounded split-width by Aiswarya [6]. In these settings, the storage mechanism is represented as a class of possible matching relations on the positions of a computation. Then, under the assumption that the resulting *behavior graphs* have bounded tree-width or split-width, respectively, there are general decidability results. In particular, decidability of scope-bounded reachability in multi-pushdown systems has been deduced via tree-width [26] and via split-width [7]. Different from underapproximations based on bounded tree-width or split-width, our framework includes multi-counter systems (such as VASS or integer VASS), but also counters nested in stacks. While VASS can be seen as special cases of multi-pushdown systems, our framework allows us, e.g. to study the complexity of scope-bounded reachability if the storage mechanism is restricted to multi-counters. On the other hand, while tree-width and split-width can be considered for queues [18, 1], they cannot be realized as storage mechanisms in valence systems.

Furthermore, after their introduction [23] scope-bounded multi-pushdown systems have been studied in terms of accepted languages [24], temporal logic model checking [26, 3]. Moreover, scope-boundedness has been studied in the timed setting [2],[4].

Over the last decade, the framework of valence automata over graph monoids has been used to study how several types of analysis are impacted by the choice of storage mechanism. For example: For which storage mechanisms (i) can silent transitions be algorithmically eliminated? [27]; (ii) do we have a Parikh's theorem [5], (iii) is (general) reachability decidable [31]; (iv) is first-order logic with reachability decidable? [10]; (v) can downward closures be computed effectively? [28].

Details of all proofs can be found in the full version of the paper.

2 Preliminaries

In this section, we recall the basics of valence systems over graph monoids [29].

Graph Monoids. This class of monoids accommodate a variety of storage mechanisms. They are defined by undirected graphs without parallel edges $\Gamma = (V, I)$ where V is a finite set of vertices and $I \subseteq \{e \subseteq V \mid 1 \leq |e| \leq 2\}$ is a finite set of undirected edges, which can be self-loops. Thus, if $\{v\} \in I$, we say that v is *looped*; otherwise, v is *unlooped*. The edge relation is also called an *independence relation*. We also write uIv for $\{u, v\} \in I$. A subset $U \subseteq V$ is a *clique* if uIv for any two distinct $u, v \in U$. If in addition, all $v \in U$ are looped, then U is a *looped clique*. If U is a clique and all $v \in U$ are unlooped, then U is an *unlooped clique*. We say that $U \subseteq V$ is an *anti-clique* if we do not have uIv for any distinct $u, v \in U$. Given the graph, we define a monoid as follows. We have the alphabet $X_\Gamma = \{v^+, v^- \mid v \in V\}$, where we write xIy for $x, y \in X_\Gamma$ if for some $u, v \in V$, we have $x \in \{u^+, u^-\}$, $y \in \{v^+, v^-\}$, $x \neq y$, and uIv. Moreover, $\equiv_\Gamma$ is the smallest congruence on X_Γ^* with $v^+v^- \equiv_\Gamma \varepsilon$ for $v \in V$ and $xy \equiv_\Gamma yx$ for xIy. Here, ε denotes the empty word. Thus, if v has a self-loop, then $v^-v^+ \equiv_\Gamma \varepsilon$. We define the monoid $\mathbb{M}\Gamma := X_\Gamma^* / \equiv_\Gamma$.

Valence Systems. Graph monoids are used in valence systems, which are finite automata whose edges are labeled with elements of a monoid. Then, a run is considered valid if the product of the monoid elements is the neutral element. Here, we only consider the case where the monoid is of the form $\mathbb{M}\Gamma$, so we define the concept directly for graphs.

Given a graph Γ, a valence system $\mathcal{A}$ over Γ consists of a finite set of states Q, and a finite transition relation $\rightarrow \subseteq Q \times X_\Gamma^* \times Q$. A *configuration* of $\mathcal{A}$ is a tuple (q, w) where $q \in Q$, $w \in X_\Gamma^*$ is the sequence of storage operations executed so far. From a configuration (q_1, u), on a transition $q_1 \xrightarrow{v} q_2$, we reach the configuration (q_2, uv). A run of $\mathcal{A}$ is a sequence of transitions. The *reachability problem* in valence systems is the following: Given states q_{init} and q_{fin}, is there a run from (q_{init}, ε) that reaches (q_{fin}, w) for some $w \in X_\Gamma^*$ with $w \equiv_\Gamma \varepsilon$?

Many classical storage types can be realized with graph monoids. Consider $\bar{\Gamma}_3 = (V, I)$ in Figure 1. We have $I = \{\{a, c\}, \{b, c\}, \{c\}\}$. For $w \in X_{\bar{\Gamma}_3}^*$ we have $w \equiv_{\bar{\Gamma}_3} \varepsilon$ if and only if two conditions are met: First, if we project to $\{a^+, a^-, b^+, b^-\}$, then the word corresponds to a sequence of push- and pop-operations that transform the empty stack into the empty stack. Here, x^+ corresponds to pushing x, and x^- to popping x, for $x \in \{a, b\}$. Second, the number of c^+ is the same as the number of c^- in w. Thus, valence automata over $\bar{\Gamma}_3$ can be seen as pushdown automata that have access to a $\mathbb{Z}$-valued counter. Similarly, the storage mechanism of Γ_2 in Figure 1 is a stack, where each stack entry is not a letter, but contains two $\mathbb{N}$-valued counters. A push (c^+) starts a new stack entry and a pop (c^-) is only possible if the topmost two counters are zero. For more examples and explanation, see [30].

▶ **Example 2.1** (Example storage mechanisms). Let us mention a few particular (classes of) graphs and how they correspond to infinite-state systems. In the following, the *direct product* of two graphs Γ and Δ is the graph obtained by taking the disjoint union of Γ and Δ and adding an edge between each vertex from Γ and each vertex from Δ.

Pushdown For $s \in \mathbb{N}$, let P_s be the graph on s vertices without edges. Then valence automata over P_s correspond to pushdown systems with s stack symbols.

Multi-pushdown Let $\mathsf{MP}_{r,s}$ be the direct product of r disjoint copies of P_s. Then valence systems over $\mathsf{MP}_{r,s}$ correspond to multi-pushdown systems with r stacks, each of which has s stack symbols. In Figure 1, the induced subgraph of graph Γ_1 on $\{b_1, b_2, b_3, c_1, c_2, c_3\}$ represents $\mathsf{MP}_{2,3}$.

VASS If UC_d is an unlooped clique with d vertices, then valence systems over UC_d correspond to d-dimensional vector addition systems with states.

Integer VASS If LC_d is a looped clique with d vertices, then valence systems over LC_d correspond to d-dimensional integer VASS.

Pushdown VASS If UC_d^- is the graph obtained from UC_{d+2} by removing a single edge, then valence systems over UC_d^- correspond to d-dimensional pushdown VASS.

3 Scope-bounded runs in valence systems

In this section, we introduce our notion of bounded scope to valence systems over arbitrary graph monoids. For each of the used concepts, we will explain how they relate to the existing notion of scope-boundedness for multi-pushdown systems. Fixing $\Gamma = (V, I)$ as before, first we introduce some preliminary notations and definitions.

Dependent sets and contexts. Recall that valence systems over the graph $\mathsf{MP}_{r,s}$ realize a storage consisting of r pushdowns, each with s stack symbols. The graph $\mathsf{MP}_{r,s}$ is a direct product of r-many disjoint anti-cliques, each with s vertices. Here, each anti-clique corresponds to a pushdown with s stack symbols: For a vertex v in such an anti-clique, the symbol v^+ is the push operation for this stack symbol, and v^- is its pop operation.

In a multi-pushdown system, a run is naturally decomposed into contexts, where each context is a sequence of operations belonging to one stack. In [19], the notion of context was generalized to valence systems as follows. A set $U \subseteq V$ is called *dependent* if it does not contain distinct vertices $u_1, u_2 \in V$ such that $u_1 I u_2$. A set of operations $Y \subseteq X_\Gamma$ is *dependent* if its underlying set of vertices $\{v \in V \mid v^+ \in Y \text{ or } v^- \in Y\}$ is dependent. A computation is called *dependent* if the set of operations occurring in it is dependent. A dependent computation is also called a *context*. In Γ_1 of Figure 1, contexts can be formed over $\{b_1, b_2, b_3\}$, $\{c_1, c_2, c_3\}$ and $\{a\}$.

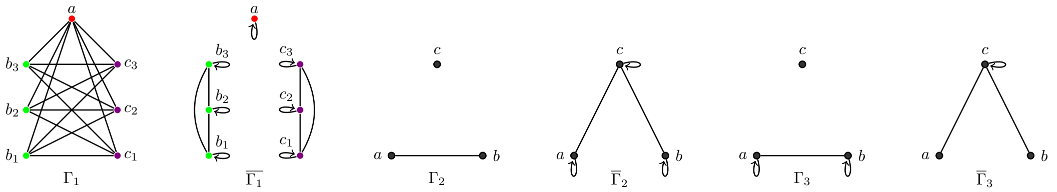


Figure 1 The storage mechanism of Γ_1 is 2 stacks and one partially blind counter. Symbols of the same stack are weakly dependent. In the storage mechanism of Γ_2, a, b, c are weakly dependent.

Context decomposition. Note that a word $w \in X_\Gamma^*$ need not have a unique decomposition into contexts. For example, for Γ_2 in Figure 1, the word $a^+c^+b^+$ can be decomposed as $(a^+c^+)b^+$ and as $a^+(c^+b^+)$. Therefore, we now define a canonical decomposition into contexts, which decomposes the word from left to right. Formally, the *canonical context decomposition* of a computation $w \in X_\Gamma^+$ (that is, $|w| > 0$) is defined inductively. If w is over a dependent set of operations, then w is a single context. Otherwise, find the maximal, non-empty prefix w_1 of w over a dependent set of operations. The canonical decomposition of w into contexts is then $w = w_1w_2 \dots w_m$ where $w_2 \dots w_m$ is the decomposition of the remaining word into contexts. In the following, unless explicitly specified otherwise, when we mention the contexts of a word, we always mean those in the canonical decomposition. Observe that in the case of $\mathsf{MP}_{r,s}$, this is exactly the decomposition into contexts of multi-pushdown systems.

Reductions. Given a computation $w = a_1 \cdots a_n$ where each $a_i \in X_\Gamma$, we identify each operation a_i with its position. We denote by $w[i]$ the ith operation of w, hence $w[i] = a_i$. A *reduction* of w is a finite sequence of applications of the following rewriting rules.

(R1) $w'.w[x].w[y].w'' \mapsto_{red} w'w''$, applicable if $w[x] = o^+, w[y] = o^-$ for some o.

(R2) $w'.w[x].w[y].w'' \mapsto_{red} w'w''$, applicable if $w[x] = o^-, w[y] = o^+$ for some oIo.

(R3) $w'.w[x].w[y].w'' \mapsto_{red} w'w[y]w[x]w''$, applicable if $w[x]Iw[y]$.

Reducing a word u to a word v using these rules is denoted by $u \mapsto_{red}^* v$. A reduction of $w = a_1 \dots a_n \in X_\Gamma^*$ to ε is the same as the free reduction of the sequence $a_1, a_2, \dots, a_n$. For any computation $w \in X_\Gamma^*$, we have $w \equiv_\Gamma \varepsilon$ iff w admits a reduction to ε [29, Equation (8.2)].

Assume that $\pi = w \mapsto_{red}^* \varepsilon$ is a reduction that transforms w into ε. The relation R_π relates positions of w which cancel in π:

$$w[x]R_\pi w[y] \ \text{ if } \ w'.w[x].w[y].w'' \mapsto_{red} w'.w'' \ \text{ or } \ w'.w[y].w[x].w'' \mapsto_{red} w'.w'' \ \text{ is used in } \ \pi$$

Greedy reductions. A word $w \in X_\Gamma^*$ is called *irreducible* if neither of the rules **R1** and **R2** is applicable in w. A reduction $\pi\colon w \mapsto_{red}^* \varepsilon$ is called *greedy* if it begins with a sequence of applications of **R1** and **R2** for each context so that the resulting context is irreducible. Note that every word w with $w \equiv_\Gamma \varepsilon$ has a greedy reduction: One can first (greedily) apply **R1** and **R2** until each context is irreducible. Since the resulting word w' still satisfies $w' \equiv_\Gamma \varepsilon$, there exists a reduction $w' \mapsto_{red}^* \varepsilon$. In total, this yields a greedy reduction.

Weak dependence. In the case of $\Gamma = \mathsf{MP}_{r,s}$, we know that any two vertices u, v are either dependent (i.e. belong to the same pushdown) or Γ is the direct product of graphs Γ_u and Γ_v such that u belongs to Γ_u and v belongs to Γ_v. This means, two operations that are not dependent can, inside every computation, be moved past each other without changing the effect on the stacks. This is not the case in general graphs. In Γ_2 in Figure 1, the vertices a and b are not dependent, but in the computation acb, they cannot be moved past each other, because none of them commutes with c. We therefore need the additional notion of weak dependence. We say that two vertices $u, v \in V$ are *weakly dependent* if there is a path between them in the complement of the graph. Here, the *complement* of a graph $\Gamma = (V, I)$ is obtained by complementing the independence relation ($v_1 I v_2$ in Γ iff we do not have $v_1 I v_2$ in the complement of Γ). Equivalently, u and v are not weakly dependent if Γ is the direct product of graphs Γ_u and Γ_v such that u belongs to Γ_u and v belongs to Γ_v. As observed above, Γ_2 shows that in general, weakly dependent vertices need not be dependent.

It can be seen that weak dependence is an equivalence relation on the set of vertices V, where the equivalence classes are the connected components in the complement of Γ. Note that all operations inside a context must belong to the same weak dependence class. We therefore say that two contexts c_1, c_2 are *weakly dependent* if their operations belong to the same weakly dependent equivalence class. Equivalently, two contexts are weakly dependent if all their letters are pairwise weakly dependent. In particular, weak dependency is an equivalence relation on contexts also. Let us denote the weak dependence equivalence relation by $\sim_W$ and by $[\ ]_{\sim_W}$ the set of all equivalence classes induced by $\sim_W$.

Scope bounded runs. We now define the notion of bounded scope computations. We first phrase the classical notion[1] of scope-boundedness [25] in our framework. If $\Gamma = \mathsf{MP}_{r,s}$, then $w \in X_\Gamma^*$ is considered k-scope bounded if there is a reduction π for w such that in between any two symbols $w[i]$ and $w[j]$ related in R_π, at most k contexts visit the same anti-clique of $w[i]$ and $w[j]$. Note that in $\mathsf{MP}_{r,s}$, for every reduction, there is a greedy reduction that induces the same relation R_π. Indeed, any applications of **R1** and **R2** that are applicable in a context at the start will eventually be made anyway: In $\mathsf{MP}_{r,s}$, if a word reduces to ε, then every position has a uniquely determined "partner position" with which is cancels in every possible reduction. Therefore, we generalize scope boundedness as follows[2].

▶ **Definition 3.1** (Scope Bounded Computations). *Consider a computation $w \in X_\Gamma^+$. We say w is k-scoped if there is a greedy reduction $\pi = w \mapsto_{red}^* \varepsilon$ such that in between any two symbols $w[i]$ and $w[j]$ related by R_π, at most $k-1$ contexts between $w[i]$ and $w[j]$ belong to the same weak dependence class as $w[i]$ and $w[j]$.*

By $\mathsf{sc}(w)$, we denote the smallest number k so that w is k-scoped. Note that there is such a k if and only if $w \equiv_\Gamma \varepsilon$. Thus, if $w \not\equiv_\Gamma \varepsilon$, we set $\mathsf{sc}(w) = \infty$. In the example in Figure 1 (graph Γ_1) the computation $w = b_1^+(c_2^+a^+c_1^+a^+c_1^-a^-c_2^-a^-)^m b_1^-$ is 3-scoped for all values of m, even though the number of context switches grows with m.

Interaction distance. We make the notion of scope bound more formal using the notion of interaction distance. Given a computation $w \in X_\Gamma^+$. Let $c_1c_2\ldots c_n$ be the canonical decomposition of w into contexts. We say that two contexts c_i, c_j with $i < j$ have an *interaction distance* K if there are $K-1$ contexts between c_i and c_j which are weakly dependent with c_i. Consider the computation $b_1^+(a^+c_1^+)^{m_1}b_2^+(a^+c_2^+)^{m_2}b_3^+(a^+c_3^+)^{m_3}b_3^-(c_3^-a^-)^{m_3}b_2^-(c_2^-a^-)^{m_2}b_1^-(c_1^-a^-)^{m_1}$. Each differently colored sequence is a context. The interaction distance between b_1^+ and b_1^- is 5, since the weakly dependent contexts strictly between them are $b_2^+, b_3^+, b_3^-, b_2^-$.

[1] The conference version [22] contains a slightly more restrictive definition. We follow the journal version [25].

[2] Another natural notion of scope-boundedness can be obtained by dropping the greediness condition. Hence, we would ask for a reduction π such that between any two R_π-related positions, there are at most $k-1$ contexts in the same weak dependence class. We expect that with this notion, Theorems 4.1 and 4.3 would still hold, but this would require changes to the algorithms.

Thus, w is k-scoped if and only if there is a greedy reduction $\pi\colon w \mapsto_{red} \varepsilon$ such that whenever $w[i]R_\pi w[j]$, then the contexts of $w[i]$ and $w[j]$ have interaction distance at most k.

The following is the central decision problem studied in this paper.

The Bounded Scope Reachability Problem(BSREACH)
Given: Graph Γ, scope bound k, valence system $\mathcal{A}$ over Γ, initial state q_{init}, final state q_{fin}
Decide: Is there a run from (q_{init}, ε) to (q_{fin}, w), for some $w \in X_\Gamma^*$ with $\mathsf{sc}(w) \leq k$?

Thus, in BSREACH, both Γ and k are part of the input. We also consider versions where certain parameters are fixed: If Γ is fixed, we denote the problem by BSREACH(Γ). If Γ is part of the input, but can be drawn from a class $\mathcal{G}$ of graphs, we write BSREACH($\mathcal{G}$). Finally, if we fix k, we use a subscript k, resulting in the problems BSREACH$_k$, BSREACH$_k(\Gamma)$, BSREACH$_k(\mathcal{G})$.

Deciding whether there is a run (q_{init}, ε) to (q_{fin}, w) with $w \equiv_\Gamma \varepsilon$ corresponds to general configuration reachability [28]. Hence, we consider the scope-bounded version of configuration reachability.

Strongly Induced Subgraphs. When we study decision problems for valence systems over graph monoids, then typically, if Δ is an induced subgraph of Γ, then a problem instance for Δ can trivially be reduced to an instance over Γ. Here, induced subgraph means that Δ can be embedded into Γ so that there is an edge in Δ iff there is one in Γ.

This is not necessarily the case for BSREACH: An induced subgraph might decompose into different weak dependence classes than Γ. Therefore, we use a stronger notion of embedding. We say that $\Gamma' = (V', I')$ is a *strongly induced subgraph* of $\Gamma = (V, I)$ if there is an injective map $\iota\colon V \to V'$ such that for any $u, v \in V$, we have (i) uIv iff $\iota(u)I'\iota(v)$ and (ii) $u \sim_W v$ iff $\iota(u) \sim_W \iota(v)$. For example, the graph Γ consisting of two adjacent vertices (without loops) is an induced subgraph of Γ_2 in Figure 1. However, Γ is not a strongly induced subgraph of Γ_2: In Γ_2, a and b are weakly dependent, whereas the vertices of Γ are not.

Neighbor Antichains. Let $\Gamma = (V, I)$ be a graph. In our algorithms, we will need to store information about a dependent set $U \subseteq V$ from which we can conclude whether for another dependent set $U' \subseteq V$, we have UIU'; that is, for all $u \in U, u' \in U', uIu'$. To estimate the required information, we use the notion of neighbor antichains. Let $\Gamma = (V, I)$ be a graph. Given $v \in V$, let $N(v)$ represent the neighbors of v, that is $N(v) = \{u \in V \mid uIv\}$. We define a quasi-ordering on V as follows. For $u, v \in V$, we have $u \leq v$ if $N(u) \subseteq N(v)$. It is possible that for distinct, $u, v \in V$ we have $u \leq v$ and $v \leq u$ and thus $\leq$ is not necessarily a partial order. In the following, we will assume that the graphs Γ are always equipped with some linear order $\ll$ on V. For example, one can just take the order in which the vertices appear in a description of Γ. Using $\ll$, we can turn $\leq$ into a partial order, which is easier to use algorithmically: We set $u \preceq v$ if and only if $u \leq v$ and $u \ll v$.

Now, given $U \subseteq V$, let $\min U = \{u \in U \mid \forall v \in U \setminus \{u\}, v \not\preceq u\}$ and $\max U = \{u \in U \mid \forall v \in U \setminus \{u\}, u \not\preceq v\}$ denote the minimal and maximal elements of U, respectively.

▶ **Lemma 3.2.** *For sets $U, U' \subseteq V$, UIU' if and only if* $(\min U)I(\min U')$.

Since $\min U$ and $\min U'$ are antichains w.r.t. $\preceq$, if we bound the size of such antichains in our graph Γ, we bound the amount of information needed to store to determine whether UIU'. We call a subset $A \subseteq V$ a *neighbor antichain* if (i) A is dependent (i.e. an anti-clique, no edges between any two vertices of A) and (ii) A is an antichain with respect to $\preceq$. For the graph Γ_1 in Figure 1, each vertex is a neighbor antichain, while for $\overline{\Gamma}_1$, each $\{a, b_i, c_j\}$ is a neighbor antichain for all i, j. By $\tau(\Gamma)$, we denote the maximal size of a neighbor antichain in Γ. Thus $\tau(\Gamma_1) = 1, \tau(\overline{\Gamma}_1) = 3$. We say that a class $\mathcal{G}$ of graphs is *neighbor antichain bounded* if there is a number t such that $\tau(\Gamma) \leq t$ for every graph Γ in $\mathcal{G}$.

For example, the class of graphs $\mathcal{G}$ consisting of bipartite graphs B_n with nodes $\{u_i, v_i \mid i \in \{1, \dots, n\}\}$, where $\{u_i, v_j\}$ is an edge iff $i \neq j$, is not neighbor antichain bounded.

4 Main results

In this section, we present the main results of this work. If both the graph and the scope bound k are part of the input, the bounded scope reachability problem is PSPACE-complete (as we will show in Theorem 4.1). Since graph monoids provide a much richer class of storage mechanisms than multi-pushdowns, this raises the question of how the complexity is affected if the storage mechanism (i.e. the graph) is drawn from a subclass of all graphs.

▶ **Theorem 4.1** (Scope bound in input). *Let $\mathcal{G}$ be a class of graphs. Then* BSREACH($\mathcal{G}$) *is*

1. NL-*complete if the graphs in $\mathcal{G}$ have at most one vertex,*
2. P-*complete if every graph in $\mathcal{G}$ is an anti-clique and $\mathcal{G}$ contains a graph with ≥ 2 vertices,*
3. PSPACE-*complete otherwise.*

▶ **Corollary 4.2.** *Let Γ be a graph. Then* BSREACH(Γ) *is*

1. NL-*complete if Γ has at most one vertex,*
2. P-*complete if Γ is an anti-clique with ≥ 2 vertices,*
3. PSPACE-*complete otherwise.*

Fixed scope bound. We notice that the problem BSREACH($\mathcal{G}$) is below PSPACE only for severely restricted classes $\mathcal{G}$, where bounded scope reachability degenerates into ordinary reachability in pushdown automata or one-counter automata. Therefore, we also study the setting where the scope bound k is fixed. However, our result requires two assumptions on the graph class $\mathcal{G}$. The first assumption is that $\mathcal{G}$ be closed under taking strongly induced subgraphs. This just rules out pathological exceptions: otherwise, it could be that there are hard instances for BSREACH$_k$ that only occur embedded in extremely large graphs in $\mathcal{G}$, resulting in lower complexity. In other words, we restrict our attention to the cases where an algorithm for $\mathcal{G}$ also has to work for strongly induced subgraphs. For each individual graph, this is always the case: if Δ is a strongly induced subgraph of Γ, then BSREACH$_k(\Delta)$ trivially reduces to BSREACH$_k(\Gamma)$.

Our second assumption is that $\mathcal{G}$ be neighbor antichain bounded. This is a non-trivial assumption that still covers many interesting types of infinite-state systems from the literature. For example, every graph mentioned in Example 2.1 has neighbor antichains of size at most 1. In particular, our result still generalizes the case of multi-pushdown systems.

Moreover, consider the graphs SC_m for $m \in \mathbb{N}$, where (i) SC_0 is a single unlooped vertex, (ii) SC_{2m+1} is obtained from SC_{2m} by adding a new vertex adjacent to all existing vertices, and (iii) SC_{2m+2} is obtained from SC_{2m+1} by adding an isolated unlooped vertex. Then neighbor antichains in SC_m are of size at most 1. Furthermore, using reductions from [31, Proposition 3.6], it follows that whenever reachability for valence systems over Γ is decidable, then this problem reduces in polynomial time to reachability over some SC_m. Whether reachability is decidable for the graphs SC_m remains an open problem [31]. Thus the graphs SC_m form an extremely expressive class that is still neighbor antichain bounded.

▶ **Theorem 4.3** (Fixed scope bound). *Let $\mathcal{G}$ be closed under strongly induced subgraphs and neighbor antichain bounded. For every $k \geq 1$, the problem* $\mathsf{BSREACH}_k(\mathcal{G})$ *is*

1. NL*-complete if $\mathcal{G}$ consists of cliques of bounded size,*
2. P*-complete if $\mathcal{G}$ contains some graph that is not a clique, and the size of cliques in $\mathcal{G}$ is bounded,*
3. PSPACE*-complete otherwise.*

In Theorem 4.3, we do not know if one can lift the restriction of neighbor antichain boundedness. In Section 8, we describe a class of graphs that is closed under strongly induced subgraphs, but we do not know the exact complexity of $\mathsf{BSREACH}_k(\mathcal{G})$.

Theorem 4.3 allows us to deduce the complexity of $\mathsf{BSREACH}_k(\Gamma)$ for every Γ.

▶ **Corollary 4.4.** *Let Γ be a graph. Then for every $k \geq 1$, the problem* $\mathsf{BSREACH}_k(\Gamma)$ *is*

1. NL*-complete if Γ is a clique,*
2. P*-complete otherwise.*

Proof. Apply Theorem 4.3 to the class consisting of Γ and its strongly induced subgraphs. ◀

Discussion of results. In the case of multi-pushdown systems, La Torre, Napoli, and Parlato [25] show that scope-bounded reachability belongs to PSPACE, and is PSPACE-hard if either the number of stacks or the scope bound k is part of the input. Our results complete the picture in several ways. If k is part of the input, then PSPACE-hardness even holds if we have two $\mathbb{N}$-valued counters instead of stacks (Theorem 4.1). Moreover, hardness also holds when we have two $\mathbb{Z}$-valued counters (which often exhibit lower complexities [14]). Moreover, we determine the complexity the case that both k and the number s of stacks is fixed.

Our results can also be interpreted in terms of vector addition systems with states (VASS). In the case of VASS (i.e. unlooped cliques), our results imply that scope-bounded reachability is PSPACE-complete if either (i) the number d of counters or (ii) the scope-bound k are part of the input (and $d \geq 2$). The same is true if we have integer VASS [14] (looped cliques).

Thus, for VASS, scope-bounding reduces the complexity of reachability from at least non-elementary [8] to PSPACE. Interestingly, for two counters, the complexity goes up from NL for general reachability [11] to PSPACE. For integer VASS, we go up from NP for general reachability [14] and for a fixed number of counters even from NL [13], to PSPACE.

Note that we obtain a much more complete picture compared to what is known for bounded context switching [19]. There, even the complexity for many individual graphs is not known. Moreover, the case of fixed context bounds has not been studied in the case of bounded context switching.

5 Block decompositions

In this section, we lay the foundation for our decision procedure in Section 6. We show that in every scope-bounded run w, each context can be decomposed into a bounded number of "blocks", which will guarantee that w can be reduced to ε by way of "block-wise" reductions. In our algorithms, this will allow us to abstract from each block (which can have unbounded length) by a finite amount of data. This is similar to the block decomposition in [19].

Let $w \in X_\Gamma^*$ such that $w \equiv_\Gamma \varepsilon$ with a reduction π. We call a decomposition $w = w_1 \cdots w_m$ a *block decomposition* if it refines the canonical context decomposition[3] and for each w_i, there is a w_j such that R_π relates every position in w_i with a position in w_i itself or in w_j. Here, we do not rule out $i = j$: A block may itself reduce to ε.

[3] In other words, each context in w consists of a contiguous subset of the factors $w_1, \ldots, w_m$.

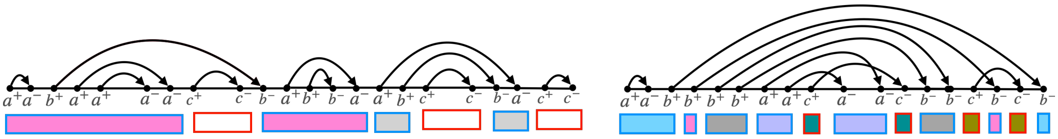

$a^+ a^- b^+ a^+ a^+ a^- a^- c^+ c^- b^- a^+ b^+ b^- a^- a^+ b^+ c^+ c^- b^- a^- c^+ c^-$
$a^+ a^- b^+ b^+ b^+ b^+ a^+ a^+ c^+ a^- a^- c^- b^- b^- c^+ b^- c^- b^-$

Figure 2 Context and block decomposition for a computation over $\overline{\Gamma}_2$ in Figure 1. The blue and red lined rectangles are the two contexts. The color filled rectangles represent blocks, with partner cancelling blocks having the same color filling.

Free reductions. Block decompositions are closely related to free reductions. Let $w_1, \ldots, w_m$ be a sequence of computations in X_Γ^*. A *free reduction* is a finite sequence of applications of the rewriting rules below to consecutive entries of the sequence so that $w_1, \ldots, w_m$ gets transformed into the empty sequence.

(FR1) $w_i, w_j \mapsto_{free} \varepsilon$ if $w_i w_j \equiv_\Gamma \varepsilon$
(FR2) $w_i, w_j \mapsto_{free} w_j, w_i$ if $w_i I w_j$
(FR3) $w_i \mapsto_{free} \hat{w}_i$ if $w_i \overset{*}{\mapsto}_{red} \hat{w}_i$ using rules **R1** and **R2**

We say that $w_1, \ldots, w_n$ is *freely reducible* if it admits a free reduction to the empty sequence.

As in [19], we have:

▶ **Proposition 5.1.** *If the decomposition $w = w_1 \cdots w_m$ refines the context decomposition, then it is a block decomposition if and only if the sequence $w_1, \ldots, w_m$ is freely reducible.*

The main result of this section is that in a scope-bounded run, there exists a block decomposition with a bounded number of factors in each context.

▶ **Theorem 5.2.** *Let $w \in X_\Gamma^*$ with $\mathsf{sc}(w) \leq k$. Then, there exists a block decomposition of w such that each context splits into at most $2k$ blocks.*

Let us sketch the proof. The block decomposition is obtained by scanning each context c from left to right. As long as there is another context c' such that all symbols either cancel inside c or with a symbol in c', we add symbols to the current block. When we encounter a symbol that cancels with a position outside of c and c', we start a new block. We show that this yields a block decomposition (see Figure 2 for an example) and with arguments similar to [19], one can show that it results in at most $2k$ blocks per context.

6 Decision procedure

In this section, we present the algorithms for the upper bounds of Theorems 4.1 and 4.3.

Block abstractions. The algorithm for bounded context switching in [19] abstracts each block by a non-deterministic automaton. This approach uses polynomial space per block, which would not be a problem for our PSPACE algorithm. However, for our P and NL algorithms, this would require too much space. Therefore, we begin with a new notion of "block abstraction", which is more space efficient.

Let $w = w_1 \cdots w_m$ be a block decomposition for a run of a valence system $\mathcal{A}$. Then it follows from Proposition 5.1 that there are words $\hat{w}_1, \ldots, \hat{w}_m$ such that $w_i \overset{*}{\mapsto}_{red} \hat{w}_i$ for each i such that the sequence $\hat{w}_1, \ldots, \hat{w}_m$ can be reduced to the empty sequence using the rewriting rules **FR1** and **FR2**. For each i, we store (i) the states occupied at the beginning and end of w_i, (ii) its first operation $f \in X_\Gamma$ in w_i, (iii) a non-deterministically chosen operation $o \in X_\Gamma$ occurring in w_i, and (iv) sets U_i^{min} and U_i^{max}, such that every maximal vertex occurring in w_i

is contained in U_i^{max} and every minimal vertex occurring in $\hat{w}_i$ is contained in U_i^{min}. In this case, we will say that the block abstraction "represents" the word $\hat{w}_i$. Thus, a *block abstration* is a tuple $N = (q_1, q_2, f, o, U^{\mathsf{min}}, U^{\mathsf{max}})$, where q_1, q_2 are states in $\mathcal{A}$, $f, o \in X_\Gamma$ are symbols, and $U^{\mathsf{min}}, U^{\mathsf{max}} \subseteq V$ are neighbor antichains. Note that for every set $B \subseteq V$, the sets $\mathsf{min}\, B$ and $\mathsf{max}\, B$ are neighbor antichains. Formally, we say that $N = (q_1, q_2, f, o, U^{\mathsf{min}}, U^{\mathsf{max}})$ *represents* $\hat{u} \in X_\Gamma^*$ if there is a word $u \in X_\Gamma^*$ such that (i) $u \mapsto_{red}^* \hat{u}$, (ii) u is read on some path from q_1 to q_2, (iii) u begins with f, (iv) o occurs in u, (iv) the set of vertices B occurring in u is a dependent set, (v) we have $\mathsf{max}\, B \subseteq U^{\mathsf{max}}$, and (vi) $\mathsf{min}\, \hat{B} \subseteq U^{\mathsf{min}}$, where $\hat{B}$ is the set of vertices occurring in $\hat{u}$. Then, $L(N)$ denotes the set of all words represented by N. We say that two block abstractions $N = (q_1, q_2, f, o, U^{\mathsf{min}}, U^{\mathsf{max}})$ and $N' = (q_1', q_2', f', o', U'^{\mathsf{min}}, U'^{\mathsf{max}})$ are *dependent* if $U^{\mathsf{max}} \cup U'^{\mathsf{max}}$ is a dependent set.

Context abstractions. Similarly, we will also need to abstract contexts. For this, we need to abstract each of its blocks. In addition, we need to store the whole context's first symbol (f) and some non-deterministically chosen other symbol (o). These additional symbols will be used to verify that we correctly guessed the canonical context decomposition of w. Thus, a *context abstraction* is a tuple $\mathcal{C} = (N_1, \ldots, N_{2k}, f, o)$, where $N_1, \ldots, N_{2k}$ are pairwise dependent block abstractions, and f and o are symbols. We say that a context abstraction $\mathcal{C} = (N_1, \ldots, N_{2k}, f, o)$ is *independent* with a context abstraction $\mathcal{C}' = (N_1', \ldots, N_{2k}', f', o')$ if (i) $f' I o$ and (ii) for some $i \in \{1, \ldots, 2k\}$, the block abstraction $N_i = (q_1^i, q_2^i, f_i, o_i, U_i^{\mathsf{min}}, U_i^{\mathsf{max}})$ satisfies $o = o_i$. Intuitively, this means if we have a word represented by $\mathcal{C}'$ and then append a word represented by $\mathcal{C}$, then these words will be the contexts in the canonical context decomposition.

In our algorithms, we will need to check whether a block decomposition admits a free reduction. This means, we need to check whether the words represented by block abstractions can cancel (to apply rule FR1) or commute (to apply FR2). Let us see how to do this. We say that the block abstractions $N = (q_1, q_2, f, o, U^{\mathsf{min}}, U^{\mathsf{max}})$ and $N' = (q_1', q_2', f', o', U'^{\mathsf{min}}, U'^{\mathsf{max}})$ *commute* if $U^{\mathsf{min}} I U'^{\mathsf{min}}$. Note that if N and N' commute, then uIu' for every $u \in L(N)$ and $u' \in L(N')$. We need an analogous condition for cancellation. We say that N and N' *cancel* if there are words $u \in L(N)$ and $u' \in L(N')$ such that $uu' \equiv_\Gamma \varepsilon$. This allows us to define an analogue of free reductions on block abstractions.

▶ **Definition 6.1.** *A* free reduction *on a sequence $N_1, \ldots, N_m$ of block abstractions is a sequence of operations*
(FRA1) *$N_i, N_j \to_{free} \varepsilon$, if N_i and N_j cancel*
(FRA2) *$N_i, N_j \to_{free} N_j, N_i$, if N_i and N_j commute.*

Together with Lemma 3.2, the following lemma allows us to verify the steps in a free reduction on block abstractions.

▶ **Lemma 6.2.** *Given Γ, a valence system over Γ, and block abstractions N_1 and N_2, one can decide in* P *whether N_1 and N_2 cancel. If Γ is a clique, this can be decided in* NL.

Given block abstractions N_1 and N_2, (i) first perform saturation [19], obtaining irreducible blocks. Saturation can be implemented using reachability in a one counter automaton, known to be NL-complete [9], (ii) second, construct a pushdown automaton that is non-empty if and only if the saturated N_1 and N_2 cancel. Emptiness of pushdown automata is decidable in P. If Γ is a clique, then the pushdown automaton uses only a single stack symbol. Hence, we only need a one-counter automaton, for which emptiness is decidable in NL [9]. This yields the two upper bounds claimed in Lemma 6.2.

Reduction to a Reachability Graph. In all our algorithms, we reduce BSREACH for a valence system $\mathcal{A}$ over Γ to reachability in a finite graph $\mathcal{R}$. For the PSPACE algorithm, we argue that each node of $\mathcal{R}$ requires polynomially many bits and the edge relation can be decided in PSPACE. For the P and the NL algorithm, we argue that $\mathcal{R}$ is polynomial in size. Moreover, for the P algorithm, we compute $\mathcal{R}$ in polynomial time. For the NL algorithm, we show that the edge relation of $\mathcal{R}$ can be decided in NL.

The vertices of $\mathcal{R}$ maintain k context abstractions (hence $2k^2$ block abstractions) per weak dependence class. Let $d \leq |V|$ be the number of weak dependence classes. The idea behind this choice of vertices is to build up a computation w from left to right, by guessing k context abstractions $(\mathcal{C}_1^\gamma, \mathcal{C}_2^\gamma, \cdots, \mathcal{C}_k^\gamma)$ per equivalence class γ which forms the "current window" of w. The initial vertex consists of k tuples $(\mathcal{E}, \ldots, \mathcal{E})$ per weak dependence equivalence class, where $\mathcal{E}$ is a placeholder representing a cancelled block or an empty block. An edge is added from a vertex v_1 to a vertex v_2 in the graph when the left most context in v_1 corresponding to an equivalence class γ cancels out completely, and v_2 is obtained by appending a fresh context abstraction $\mathcal{C}_y^\gamma$ to v_1. Indeed if w is k scope bounded, then the blocks of the first context abstraction $\mathcal{C}_1^\gamma$ must cancel out with blocks from the remaining context abstractions $\mathcal{C}_2^\gamma, \cdots, \mathcal{C}_k^\gamma$ using the free reduction rules discussed above. We can guess an equivalence class γ whose context abstraction $\mathcal{C}_1^\gamma$ cancels out, and extend w by guessing the next context abstraction $\mathcal{C}_y^\gamma$ in the same equivalence class. An edge between two vertices in $\mathcal{R}$ represents an extension of w where a new context of an appropriate equivalence class is added, after the leftmost context has cancelled out using some free reduction rules.

For a weak dependence class, we refer to the tuple of k contexts of interest as a *configuration*. Thus, a vertex in $\mathcal{R}$ consists of d configurations. As discussed earlier in section 6, we represent the $2k^2$ blocks in each configuration using block abstractions.

▶ **Definition 6.3.** *Given a weak dependence class $\gamma \in [\,]_{\sim_W}$, a* configuration of γ *is a k-tuple of the form $s_\gamma = (\mathcal{C}_1^\gamma, \mathcal{C}_2^\gamma, \cdots, \mathcal{C}_k^\gamma)$, where each $\mathcal{C}_c^\gamma$ for $1 \leq c \leq k$ is a context abstraction.*

As mentioned above, in slight abuse of terminology, in case of cancellation in free reductions, we also allow $\mathcal{E}$ as a placeholder for cancelled contexts.

Intuitively, the configuration tracks the remaining non-cancelled blocks of the last k contexts of this weak dependence class along with their relative positions in their contexts.

▶ **Definition 6.4.** *A vertex in the graph $\mathcal{R}$ has the form $(s_{\gamma_1}, \ldots, s_{\gamma_d} | i, q)$ where (i) $\gamma_1, \ldots, \gamma_d$ are the distinct equivalence classes in $[\,]_{\sim_W}$, (ii) each s_{γ_j} is a configuration, (iii) $i \in \{1, \cdots, d\}$ is the current weak dependence class we are on, and (iv) q is the last state occurring in s_{γ_i}.*

Here, if s_{γ_i} consists just of $\mathcal{E}$, then the last condition is satisfied automatically.

▶ **Definition 6.5.** *For $\gamma \in [\,]_{\sim_W}$, a configuration s'_γ is* one-step reachable *from a configuration $s_\gamma = (\mathcal{C}_1^\gamma, \ldots, \mathcal{C}_k^\gamma)$ iff there exists some context abstraction $\mathcal{C}$ and a sequence of free reduction operations on the sequence of block abstractions*

$$N_1^{\gamma 1}, \ldots, N_{2k}^{\gamma 1}, N_1^{\gamma 2}, \ldots, N_{2k}^{\gamma 2}, \ldots N_1^{\gamma k}, \ldots, N_{2k}^{\gamma k}, N_1, \ldots, N_{2k}$$

resulting in the new sequence (placing $\mathcal{E}$ in a position if the block was cancelled due to the free reduction rules; otherwise we keep the same block abstraction)

$$N_1'^{\gamma 1}, \ldots, N_{2k}'^{\gamma 1}, N_1'^{\gamma 2}, \ldots, N_{2k}'^{\gamma 2}, \ldots N_1'^{\gamma k}, \ldots, N_{2k}'^{\gamma k}, N_1'^{\gamma(k+1)}, \ldots, N_{2k}'^{\gamma(k+1)}$$

such that $N_\ell'^{\gamma 1} = \mathcal{E}$, for all $\ell \in \{1, \ldots, 2k\}$, and $\left(\mathcal{C}'^\gamma_2, \mathcal{C}'^\gamma_3, \cdots, \mathcal{C}'^\gamma_k, \mathcal{C}'^\gamma_{k+1}\right) = s'_\gamma$, where $\mathcal{C}'^\gamma_\ell = (N_1'^{\gamma\ell}, N_2'^{\gamma\ell}, \cdots, N_{2k}'^{\gamma\ell}, f^{\gamma\ell}, o^{\gamma\ell})$ for $\ell \in \{1, \ldots, k+1\}$, and $N_1, \ldots, N_{2k}$ are the block abstractions in $\mathcal{C}$. In this case, we write $s_\gamma \xrightarrow{\mathcal{C}} s'_\gamma$.

In short, we can go from s_γ to s'_γ in one step if we can add some context abstraction to s_γ so that using free reduction steps, we can reach s'_γ along with clearing the oldest context.

We are now ready to define the edge relation of $\mathcal{R}$.

▶ **Definition 6.6.** *There is an edge in $\mathcal{R}$ from a vertex $v = (s_{\gamma_1}, \dots, s_{\gamma_d}|j, q)$ to a vertex v' iff there is some $i \in \{1, \dots, d\}$ and a configuration s'_{γ_i} such that (i) $s_{\gamma_i} \xrightarrow{\mathcal{C}'} s'_{\gamma_i}$ for some context abstraction $\mathcal{C}'$ such that $\mathcal{C}'$ is independent with the last context abstraction $\mathcal{C}$ of s_{γ_j} and (ii) q is the first state in $\mathcal{C}'$ and (iii) $v' = (s_{\gamma_1}, \dots, s_{\gamma_{i-1}}, s'_{\gamma_i}, s_{\gamma_{i+1}}, \dots, s_{\gamma_d}|i, q')$, where q' is the last state of $\mathcal{C}'$.*

Since a context is a maximal dependent sequence, this check suffices to guarantee independence of words represented by $\mathcal{C}'$ and $\mathcal{C}$.

As mentioned above, our algorithm reduces scope-bounded reachability to reachability between two nodes in $\mathcal{R}$. Details can be found in the full version.

Complexity. We turn to the upper bounds in Theorems 4.1 and 4.3. Asymptotically, a block abstraction requires $\log|Q| + 2t \cdot \log|V|$ bits, where t is an upper bound on the size of neighbor antichains. Per context, we store $2k$ block abstractions and two symbols. Let d be the number of weak dependence classes. In a node of $\mathcal{R}$, we store k contexts per weak dependence class, a number in $\{1, \dots, d\}$, and a state. Hence, asymptotically, we need $M = dk^2(\log|Q| + t \cdot \log|V|) + \log d + \log|Q|$ bits per node of $\mathcal{R}$. To simulate the free reduction, we only need a constant multiple of this. We can thus decide reachability in $\mathcal{R}$ in PSPACE.

▶ **Proposition 6.7.** BSREACH *is in* PSPACE*.*

We now look at the upper bounds for the first and second cases of Theorem 4.3.

▶ **Proposition 6.8.** *Let $\mathcal{G}$ be a class of graphs that is closed under strongly induced subgraphs and neighbor antichain bounded. If $\mathcal{G}$ consists of cliques of bounded size, then for each $k \geq 1$, the problem* BSREACH$_k(\mathcal{G})$ *belongs to* NL*.*

Proof. Our assumptions imply that $d \leq |V|$, t, and k are bounded. Thus M is at most logarithmic in the input. Moreover, we can simulate free reductions using logarithmic space, because checking whether two block abstractions cancel can be done in NL by Lemma 6.2. ◀

▶ **Proposition 6.9.** *Let $\mathcal{G}$ be a class of graphs that is closed under strongly induced subgraphs and neighbor antichain bounded. If the size of cliques in $\mathcal{G}$ is bounded, then for every $k \geq 1$, the problem* BSREACH$_k(\mathcal{G})$ *belongs to* P*.*

Proof. First observe that as in Proposition 6.8, the parameters d, t, and k are bounded. To see this for d, let ℓ be an upper bound on the size of cliques in $\mathcal{G}$. Then, every graph Γ in $\mathcal{G}$ can have at most ℓ weak dependence classes: Otherwise, Γ would have a clique with $\ell + 1$ nodes as a strongly induced subgraph, and thus $\mathcal{G}$ would contain a clique with $\ell + 1$ nodes. Hence, d is bounded and for a node in $\mathcal{R}$, we need only logarithmic space. Moreover, by Lemma 6.2, we can verify a free reduction step in P. ◀

Special Graphs. We turn to the NL and P upper bounds for Theorem 4.1. In each case, all graphs are anti-cliques. Thus, every run has a single context and BSREACH reduces to membership for pushdown automata, which is in P. If there is just one vertex, we can even obtain a one-counter automaton, for which emptiness is in NL [9].

▶ **Proposition 6.10.** *If $\mathcal{G}$ is the class of anti-cliques, then* BSREACH$(\mathcal{G})$ *is in* P*. Moreover, if Γ has only one vertex, then* BSREACH(Γ) *is in* NL*.*

7 Hardness

In this section, we show the hardness results of Theorems 4.1 and 4.3.

▶ **Proposition 7.1.** *If Γ is not a clique, then* $\mathsf{BSREACH}_k(\Gamma)$ *is* P*-hard for each* $k \geq 1$.

This uses standard techniques. If a valence system uses only the two non-adjacent vertices in Γ, then scope-bounded reachability is the same as ordinary reachability. If both vertices are looped, then this is the rational subset membership problem for a free group of rank 2, for which P-hardness was observed in [15, Theorem III.4]. If at least one vertex is unlooped, a standard encoding yields a reduction from emptiness of pushdown automata.

Two adjacent vertices. Our second hardness proof shows PSPACE-hardness in Theorem 4.1.

▶ **Proposition 7.2.** *If Γ has two adjacent nodes, then* $\mathsf{BSREACH}(\Gamma)$ *is* PSPACE*-hard.*

For the proof of Proposition 7.2, we employ the model of bounded queue automata. A *bounded queue automaton (BQA)* is a tuple $\mathcal{A} = (Q, n, T, q_0, q_f)$, where (i) Q is a finite set of *states*, (ii) $n \in \mathbb{N}$ is the *queue length*, given in unary, (iii) T is its set of *transitions*, (iv) $q_0 \in Q$ is its *initial state*, and (iv) $q_f \subseteq Q$ is its *final state*. A *configuration* of a BQA is a pair $(q, w) \in Q \times \{0,1\}^n$. A transition is of the form (q, x, y, q'), where $q, q' \in Q$ and $x, y \in \{0, 1\}$. We write $(q, w) \to (q', w')$ if there is a transition (q, x, y, q') such that (i) w has prefix x and (ii) removing x from the left and appending y on the right yields w'. The *reachability problem for BQA* is the following: Given a bounded queue automaton (Q, n, T, q_0, q_f), is it true that $(q_0, 0^n) \xrightarrow{*} (q_f, 0^n)$? It is straightforward to simulate a linear bounded automaton using a bounded queue automaton and vice-versa, hence reachability for BQA is PSPACE-complete.

General idea and challenge. Let us first assume that the nodes u and v in Γ are not weakly dependent. The initial approach for Proposition 7.2 is to encode the queue content in the current window of $k = n$ contexts. In each context, we encode a 0-bit using an occurrence of the letter u^+ that can only be cancelled with a future u^-. We call this a 0*-context.* Likewise, a 1-bit is encoded by two occurrences of u, which we call a 1*-context.* Therefore, we abbreviate $\mathbf{0} = u^+$ and $\mathbf{1} = u^+u^+$. We also have the right inverses $\bar{\mathbf{0}} = u^-$ and $\bar{\mathbf{1}} = u^-u^-$. To start a new context, we multiply v^+v^- and use the abbreviation $\| = v^+v^-$. With this encoding, it is easy to check that the oldest context is a 0-context: Just multiply $\bar{\mathbf{0}} = u^-$ and then start a new context using $\| = v^+v^-$. This can only succeed if the oldest context encodes a 0: If it had encoded a 1, there would be another occurrence of u^+ that can never be cancelled.

However, it is not so easy to check that the oldest context is a 1-context. One could multiply with $\bar{\mathbf{1}}\| = u^-u^-v^+v^-$, but this can succeed even if the oldest context is a 0-context: Indeed, the first occurrence of u^- can cancel with the u^+ in the oldest context c, but the second u^- could cancel with u^+ in a context to the right of c.

Solution. We overcome this as follows. Instead of one context per bit, we use three contexts. To encode a 0 in the queue, we use a 0-context, a 1-context, and another 1-context, resulting in the string 011. To encode a 1, we do the same with the bit string 100. Then, we use the above approach to check for 011 or 100: Since a successful check for a 0-context guarantees that there was a 0-context, checking for $0, 1, 1$ guarantees that the oldest context is a 0-context, thus the three oldest contexts must carry 011. When we check for $1, 0, 0$, then among the three oldest contexts, at least two are 0-contexts, hence the three oldest contexts carry 100.

Let us calculate the required scope bound to implement this idea. Since we only use the operations u^+, u^-, v^+, v^-, we only have u-contexts (consisting of u^+, u^-) and v-contexts (consisting of v^+, v^-). When we read the oldest bit in the queue, we produce three new u-contexts (separated by v-contexts). Then, we need to write a new bit in the queue, which requires another three u-contexts (separated by v-contexts). The interaction distance to the oldest u-context that is part of the leftmost queue bit is always $k = 3(2n-1)$: The oldest bit is encoded using three u-contexts. For each further queue entry $i \in \{2, \ldots, n\}$, we have three u-contexts that were used to read an even older bit, and then three u-contexts that encode the i-th bit in the queue. In total, this yields $3(1 + 2(n-1)) = 3(2n-1)$ many u-contexts.

To initialize the queue, we use $t_0 = (\mathbf{0}\|\mathbf{1}\|\mathbf{1}\|)(\mathbf{e}\|\mathbf{e}\|\mathbf{e}\|\mathbf{0}\|\mathbf{1}\|\mathbf{1}\|)^{n-1}$. Here, $\mathbf{e} = u^+u^-$ is a "gap context" that ensures that the leftmost bit has interaction distance exactly $k = 3(2n-1)$ from the right end of t_0. Thus, t_0 puts n copies of the bit string 011, plus $n-1$ gap contexts into our window of $k = 3(2n-1)$ contexts. To simulate a transition (p, x, y, p'), we check that x is the bit encoded by the three oldest contexts. Afterwards, we put the new bit y into the queue. Therefore, if $x = 0$, define the triple $(x_1, x_2, x_3) = (\bar{\mathbf{0}}, \bar{\mathbf{1}}, \bar{\mathbf{1}})$; if $x = 1$, let $(x_1, x_2, x_3) = (\bar{\mathbf{1}}, \bar{\mathbf{0}}, \bar{\mathbf{0}})$. Moreover, if $y = 0$, then let $(y_1, y_2, y_3) = (\mathbf{0}, \mathbf{1}, \mathbf{1})$; if $y = 1$, then let $(y_1, y_2, y_3) = (\mathbf{1}, \mathbf{0}, \mathbf{0})$. Then we use the string $t_{x,y} = x_1\|x_2\|x_3\|y_1\|y_2\|y_3\|$. Finally, to check that the encoded queue content consists entirely of 0's, we use $t_f = (\bar{\mathbf{0}}\|\bar{\mathbf{1}}\|\bar{\mathbf{1}}\|)(\mathbf{e}\|\mathbf{e}\|\mathbf{e}\|\bar{\mathbf{0}}\|\bar{\mathbf{1}}\|\bar{\mathbf{1}}\|)^{n-1}$. With this encoding, it is straightforward to translate a BQA into a valence system over Γ.

Note that if u and v are weakly dependent, then the same construction works, except that we have to set $k = 6(2n-1)$, because now the v-contexts $\| = v^+v^-$ between two u-contexts count towards the interaction distance.

Unbounded cliques. We turn to PSPACE-hardness in Theorem 4.3.

▶ **Proposition 7.3.** *Suppose $\mathcal{G}$ be either the class of unlooped cliques or the class of looped cliques. Then for every $k \geq 1$, the problem* $\mathsf{BSREACH}_k(\mathcal{G})$ *is* PSPACE*-hard.*

Here it is convenient to reduce from bit vector automata, whose configuration consists of a state and a bit vector. In each step, they can read and modify one of the bits. A *bit vector automaton (BVA)* is a tuple (Q, n, T, q_0, q_f), where (i) Q is a finite set of states, (ii) n is the *vector length*, given in unary, (iii) a set T of *transitions*, (iv) $q_0 \in Q$ is its *initial state*, and (v) $q_f \in Q$ is its *final state*. A transition is of the form (p, i, x, y, q) with $p, q \in Q$, $i \in \{0, \ldots, n\}$, and $x, y \in \{0, 1\}$. It checks that i-th bit is currently x, and sets the i-th bit to y. Thus, a *configuration* of a bit vector automaton is a pair $(q, w) \in Q \times \{0,1\}^n$. By $\xrightarrow{*}$, we denote the reachability relation. The *reachability problem for BVA* asks, given a BVA (Q, n, T, q_0, q_f), is it true that $(q_0, 0^n) \xrightarrow{*} (q_f, 0^n)$? Again, a simulation of linear bounded automata is straightforward and this problem is PSPACE-complete.

Proof of Proposition 7.3. Let $\mathcal{A} = (Q, n, T, q_0, q_f)$ be a BVA. Moreover, depending on whether $\mathcal{G}$ is the class of looped or unlooped cliques, let $\Gamma = (V, I)$ be either a looped or an unlooped clique with $2n$ vertices, so let $V = \{a_i, b_i \mid i \in \{1, \ldots, n\}\}$. Our construction does not depend on whether Γ is looped or unlooped and we will show that it is correct in either case. We first illustrate the idea for maintaining a single bit using the vertices a_i, b_i. To ease notation, we now write v for v^+ and $\bar{v}$ for v^- when $v \in V$. Consider the string

$$w = (a_i^{r_1} b_i \bar{b}_i \bar{a}_i^{s_1} b_i \bar{b}_i)(a_i \bar{a}_i b_i \bar{b}_i)^k (a_i^{r_2} b_i \bar{b}_i \bar{a}_i^{s_2} b_i \bar{b}_i) \cdots (a_i \bar{a}_i b_i \bar{b}_i)^k (a_i^{r_m} b_i \bar{b}_i \bar{a}_i^{s_m} b_i \bar{b}_i).$$

Moreover, assume that for each $j = 1, \ldots, m$, we have $r_j, s_j \in \{k, 3k\}$. We think of $a_i^{r_j} b_i \bar{b}_i$ as an operation that stores 0 if $r_j = k$ and stores 1 if $r_j = 3k$. We think of $\bar{a}_i^{s_j} b_i \bar{b}_i$ as a read operation, where again $s_j = k$ stands for 0 and $s_j = 3k$ stands for 1. Here, the purpose of $b_i \bar{b}_i$ is to start a new context (since Γ is a clique). Moreover, each factor $(a_i \bar{a}_i b_i \bar{b}_i)^k$ produces

k contexts in the weak dependence class of a_i, where each context contains $a_i\bar{a}_i$. This means, each factor $(a_i\bar{a}_ib_i\bar{b}_i)^k$ enforces an interaction distance of $k+1$ between $\bar{a}_i^{s_j}$ and $a_i^{r_j+1}$, and thus prevents them from canceling with each other.

We claim that $\mathsf{sc}(w) \leq k$ if and only if each read operation reads the bit that was stored before. In other words, we have $\mathsf{sc}(w) \leq k$ if and only if $r_j = s_j$ for each $j \in \{1, \ldots, m\}$. Moreover, this is true regardless of whether Γ is looped or unlooped. For the "if", note that each $a_i^{r_j}$ can cancel with $\bar{a}_i^{s_j}$ and in every other context, every letter (a_i, $\bar{a}_i$, b_i, $\bar{b}_i$) can cancel with its direct neighbor. Conversely, suppose $r_j \neq s_j$ for some j. If $r_j = 3k$ and $s_j = k$, then the context $a_i^{r_j} = a_i^{3k}$ sees only $3k-1$ occurrences of $\bar{a}_i$ in contexts at interaction distance $\leq k$: First, the context $\bar{a}_i^{s_i} = \bar{a}_i^k$ yields k occurrences. The other $2k-1$ contexts are of the form $a_i\bar{a}_i$ and each provides one occurrence of $\bar{a}_i$. In total, we have $k + 2k - 1 = 3k - 1$. It is thus impossible to cancel every letter in $a_i^{r_j}$. The case $r_j = k$ and $s_j = 3k$ is symmetric. This proves our claim. Using this encoding, it is now straightforward to simulate n bits. ◀

Propositions 7.1–7.3 complete our proofs: Theorem 4.1 follows from Propositions 7.1, 7.2, 6.7, and 6.10. Theorem 4.3 follows from Propositions 7.1, 7.3, and 6.7–6.9.

8 Conclusion

We have introduced a notion of scope-bounded reachability for valence systems over graph monoids. In the special case of graphs that correspond to multi-pushdowns, this notion coincides with the original notion of scope-boundedness introduced by La Torre, Napoli, and Parlato [25]. We have shown that with this notion, scope-bounded reachability is decidable in PSPACE, even if the graph and the scope bound k are part of the input.

In addition, we have studied the complexity of the problem under four types of restrictions: (i) k and the graph are part of the input, and the graph is drawn from some class $\mathcal{G}$ of graphs, (Theorem 4.1), (ii) k is part of the input and the graph is fixed (Corollary 4.2), (iii) k is fixed and the graph is drawn from some class $\mathcal{G}$ of graphs that is neighbor antichain bounded and closed under strongly induced subgraphs (Theorem 4.3) and (iv) k is fixed and the graph is fixed (Corollary 4.4). We have completely determined the complexity landscape in the situations (i)–(iv): In every case, we obtain NL-, P-, or PSPACE-completeness. These results settle the complexity of scope-bounded reachability for most types of infinite-state systems that fit in the framework of valence systems and have been considered in the literature.

Open Problem: Dropping neighbor antichain boundedness. We leave open what complexities can arise if in case (iii) above, we drop the assumption of neighbor antichain boundedness. In other words: k is fixed and the graph comes from a class $\mathcal{G}$ that is closed under strongly induced subgraphs. For all we know, it is possible that there are classes $\mathcal{G}$ for which the problem is neither NL-, nor P-, nor PSPACE-complete.

For example, for each $n \geq 0$, consider the bipartite graph B_n with nodes $\{u_i, v_i \mid i \in \{1, \ldots, n\}\}$, where $\{u_i, v_j\}$ is an edge if and only if $i \neq j$. Moreover, let $\mathcal{G}$ be the class of graphs containing B_n for every $n \in \mathbb{N}$ and all strongly induced subgraphs. Observe that the cliques in $\mathcal{G}$ have size at most 2: B_n is bipartite and thus every clique in B_n has size at most 2. Moreover, the graphs B_n have neighbor antichains of unbounded size: The set $\{u_1, \ldots, u_n\}$ is a neighbor antichain in B_n.

We currently do not know the exact complexity of $\mathsf{BSREACH}_k(\mathcal{G})$. By Theorem 4.3, the problem is P-hard and in PSPACE. Intuitively, our P upper bound does not apply because in each node of $\mathcal{R}$, one would have to remember n bits in order to keep enough information about commutation of blocks: For a subset $S \subseteq \{1, \ldots, n\}$, let u_S be the product of all $u_1^+, \ldots, u_n^+$, where we only include u_i^+ if $i \in S$. Then $u_S v_j^+ \equiv v_j^+ u_S$ if and only if $j \notin S$.

References

1 C. Aiswarya, Paul Gastin, and K. Narayan Kumar. Verifying communicating multi-pushdown systems via split-width. In *Automated Technology for Verification and Analysis – 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, volume 8837 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2014. doi:10.1007/978-3-319-11936-6_1.

2 S. Akshay, Paul Gastin, Shankara Narayanan Krishna, and Sparsa Roychowdhury. Revisiting underapproximate reachability for multipushdown systems. In *Tools and Algorithms for the Construction and Analysis of Systems – 26th International Conference, TACAS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings, Part I*, volume 12078 of *Lecture Notes in Computer Science*, pages 387–404. Springer, 2020. doi:10.1007/978-3-030-45190-5_21.

3 Mohamed Faouzi Atig, Ahmed Bouajjani, K. Narayan Kumar, and Prakash Saivasan. Linear-time model-checking for multithreaded programs under scope-bounding. In *Automated Technology for Verification and Analysis – 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings*, volume 7561 of *Lecture Notes in Computer Science*, pages 152–166. Springer, 2012. doi:10.1007/978-3-642-33386-6_13.

4 Devendra Bhave, Shankara Narayanan Krishna, Ramchandra Phawade, and Ashutosh Trivedi. On timed scope-bounded context-sensitive languages. In *Developments in Language Theory – 23rd International Conference, DLT 2019, Warsaw, Poland, August 5-9, 2019, Proceedings*, volume 11647 of *Lecture Notes in Computer Science*, pages 168–181. Springer, 2019. doi:10.1007/978-3-030-24886-4_12.

5 P. Buckheister and Georg Zetzsche. Semilinearity and context-freeness of languages accepted by valence automata. In *Mathematical Foundations of Computer Science 2013 – 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, volume 8087 of *Lecture Notes in Computer Science*, pages 231–242. Springer, 2013. doi:10.1007/978-3-642-40313-2_22.

6 Aiswarya Cyriac. *Verification of communicating recursive programs via split-width. (Vérification de programmes récursifs et communicants via split-width)*. PhD thesis, École normale supérieure de Cachan, France, 2014. URL: https://tel.archives-ouvertes.fr/tel-01015561.

7 Aiswarya Cyriac, Paul Gastin, and K. Narayan Kumar. MSO decidability of multi-pushdown systems via split-width. In *CONCUR 2012 – Concurrency Theory – 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, volume 7454 of *Lecture Notes in Computer Science*, pages 547–561. Springer, 2012. doi:10.1007/978-3-642-32940-1_38.

8 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for petri nets is not elementary. In *Proceedings of STOC 2019*, pages 24–33. ACM, 2019. doi:10.1145/3313276.3316369.

9 Stéphane Demri and Régis Gascon. The effects of bounding syntactic resources on presburger LTL. In *Proceedings of TIME 2007*, pages 94–104. IEEE Computer Society, 2007. doi:10.1109/TIME.2007.63.

10 Emanuele D'Osualdo, Roland Meyer, and Georg Zetzsche. First-order logic with reachability for infinite-state systems. In *Proceedings of LICS 2016*, pages 457–466. ACM, 2016. doi:10.1145/2933575.2934552.

11 Matthias Englert, Ranko Lazic, and Patrick Totzke. Reachability in two-dimensional unary vector addition systems with states is nl-complete. In *Proceedings of LICS 2016*, pages 477–484. ACM, 2016. doi:10.1145/2933575.2933577.

12 Javier Esparza, Pierre Ganty, and Tomás Poch. Pattern-based verification for multithreaded programs. *ACM Trans. Program. Lang. Syst.*, 36(3):9:1–9:29, 2014. doi:10.1145/2629644.

13 Eitan M. Gurari and Oscar H. Ibarra. The complexity of decision problems for finite-turn multicounter machines. *Journal of Computer and System Sciences*, 22(2):220–229, 1981. doi:10.1016/0022-0000(81)90028-3.

14 Christoph Haase and Simon Halfon. Integer vector addition systems with states. In *Proceedings of RP 2014*, volume 8762 of *Lecture Notes in Computer Science*, pages 112–124. Springer, 2014. doi:10.1007/978-3-319-11439-2_9.

15 Christoph Haase and Georg Zetzsche. Presburger arithmetic with stars, rational subsets of graph groups, and nested zero tests. In *Proceedings of LICS 2019*, pages 1–14. IEEE, 2019. doi:10.1109/LICS.2019.8785850.

16 Serge Haddad and Denis Poitrenaud. Recursive Petri nets. *Acta Informatica*, 44(7):463–508, 2007.

17 Jérôme Leroux, Grégoire Sutre, and Patrick Totzke. On the coverability problem for pushdown vector addition systems in one dimension. In *Proceedings of ICALP 2015*, volume 9135 of *Lecture Notes in Computer Science*, pages 324–336. Springer, 2015. doi:10.1007/978-3-662-47666-6_26.

18 P. Madhusudan and Gennaro Parlato. The tree width of auxiliary storage. In *Proceedings of POPL 2011*, pages 283–294. ACM, 2011. doi:10.1145/1926385.1926419.

19 Roland Meyer, Sebastian Muskalla, and Georg Zetzsche. Bounded context switching for valence systems. In *29th International Conference on Concurrency Theory, CONCUR 2018, September 4-7, 2018, Beijing, China*, volume 118 of *LIPIcs*, pages 12:1–12:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.12.

20 Shaz Qadeer and Jakob Rehof. Context-bounded model checking of concurrent software. In *Proceedings of TACAS 2005*, pages 93–107. Springer, 2005.

21 Ganesan Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Transactions on Programming languages and Systems (TOPLAS)*, 22(2):416–430, 2000.

22 Salvatore La Torre and Margherita Napoli. Reachability of multistack pushdown systems with scope-bounded matching relations. In *CONCUR 2011 – Concurrency Theory – 22nd International Conference, CONCUR 2011, Aachen, Germany, September 6-9, 2011. Proceedings*, volume 6901 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2011. doi:10.1007/978-3-642-23217-6_14.

23 Salvatore La Torre and Margherita Napoli. A temporal logic for multi-threaded programs. In *Theoretical Computer Science – 7th IFIP TC 1/WG 2.2 International Conference, TCS 2012, Amsterdam, The Netherlands, September 26-28, 2012. Proceedings*, volume 7604 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 2012. doi:10.1007/978-3-642-33475-7_16.

24 Salvatore La Torre, Margherita Napoli, and Gennaro Parlato. Scope-bounded pushdown languages. *Int. J. Found. Comput. Sci.*, 27(2):215–234, 2016. doi:10.1142/S0129054116400074.

25 Salvatore La Torre, Margherita Napoli, and Gennaro Parlato. Reachability of scope-bounded multistack pushdown systems. *Inf. Comput.*, 275:104588, 2020. doi:10.1016/j.ic.2020.104588.

26 Salvatore La Torre and Gennaro Parlato. Scope-bounded multistack pushdown systems: Fixed-point, sequentialization, and tree-width. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2012, December 15-17, 2012, Hyderabad, India*, volume 18 of *LIPIcs*, pages 173–184. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.173.

27 Georg Zetzsche. Silent transitions in automata with storage. In *Proceedings of ICALP 2013*, volume 7966 of *Lecture Notes in Computer Science*, pages 434–445. Springer, 2013. doi:10.1007/978-3-642-39212-2_39.

28 Georg Zetzsche. Computing downward closures for stacked counter automata. In *Proceedings of STACS 2015*, volume 30 of *LIPIcs*, pages 743–756. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015. doi:10.4230/LIPIcs.STACS.2015.743.

29 Georg Zetzsche. *Monoids as Storage Mechanisms*. PhD thesis, Kaiserslautern University of Technology, Germany, 2016. URL: https://kluedo.ub.uni-kl.de/frontdoor/index/index/docId/4400.

30 Georg Zetzsche. Monoids as storage mechanisms. *Bull. EATCS*, 120, 2016. URL: `http://eatcs.org/beatcs/index.php/beatcs/article/view/459`.

31 Georg Zetzsche. The emptiness problem for valence automata over graph monoids. *Inf. Comput.*, 277:104583, 2021. `doi:10.1016/j.ic.2020.104583`.

Deciding Polynomial Termination Complexity for VASS Programs

Michal Ajdarów
Masaryk University, Brno, Czech Republic

Antonín Kučera
Masaryk University, Brno, Czech Republic

Abstract

We show that for every fixed degree $k \geq 3$, the problem whether the termination/counter complexity of a given demonic VASS is $\mathcal{O}(n^k)$, $\Omega(n^k)$, and $\Theta(n^k)$ is **coNP**-complete, **NP**-complete, and **DP**-complete, respectively. We also classify the complexity of these problems for $k \leq 2$. This shows that the polynomial-time algorithm designed for strongly connected demonic VASS in previous works cannot be extended to the general case. Then, we prove that the same problems for VASS games are **PSPACE**-complete. Again, we classify the complexity also for $k \leq 2$. Tractable subclasses of demonic VASS and VASS games are obtained by bounding certain structural parameters, which opens the way to applications in program analysis despite the presented lower complexity bounds.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Termination complexity, vector addition systems

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.30

Related Version *Full Version*: `https://arxiv.org/abs/2102.06889` [1]

Funding The work is supported by the Czech Science Foundation, Grant No. 21-24711S.

1 Introduction

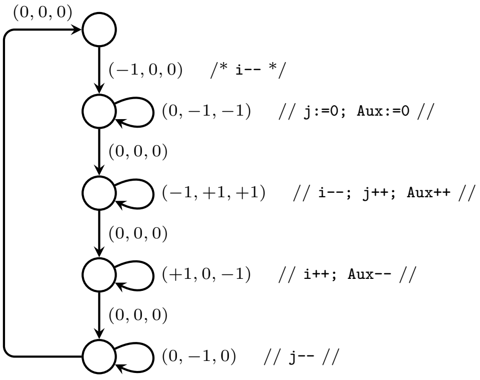
Vector addition systems with states (VASS) are a generic formalism expressively equivalent to Petri nets. In program analysis, VASS are used to model programs with unbounded integer variables, parameterized systems, etc. Thus, various problems about such systems reduce to the corresponding questions about VASS. This approach's main bottleneck is that interesting questions about VASS tend to have high computational complexity (see, e.g., [8, 15, 16]). Surprisingly, recent results (see below) have revealed computational tractability of problems related to *asymptotic complexity* of VASS computations, allowing to answer questions like "Does the program terminate in time polynomial in n for all inputs of size n?", or "Is the maximal value of a given variable bounded by $\mathcal{O}(n^4)$ for all inputs of size n?". These results are encouraging and may enhance the existing software tools for asymptotic program analysis such as SPEED [11], COSTA [2], RAML [12], Rank [3], Loopus [18, 19], AProVE [10], CoFloCo [9], C4B [7], and others. In this paper, we give a full classification of the computational complexity of deciding polynomial termination/counter complexity for demonic VASS and VASS games, and solve open problems formulated in previous works. Furthermore, we identify structural parameters making the asymptotic VASS analysis computationally hard. Since these parameters are often small in VASS program abstractions, this opens the way to applications in program analysis despite the established lower complexity bounds.

The *termination complexity* of a given VASS $\mathcal{A}$ is a function $\mathcal{L} : \mathbb{N} \to \mathbb{N}_\infty$ assigning to every n the maximal length of a computation initiated in a configuration with all counters initialized to n. Similarly, the *counter complexity* of a given counter c in $\mathcal{A}$ is a function

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 30; pp. 30:1–30:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

```
input i;
while(i>0)
   i:=i-1;
   j:=i;
   while(j>0)
      j:=j-1;
```

```
input i;
while(i>0)
   i--;
   j:=0; Aux:=0;
   while(i>0)
      i--;
      j++;
      Aux++;
   while(Aux>0)
      i++;
      Aux--;
   while(j>0)
      j--;
```



$(0,0,0)$
$(-1,0,0)$ /* i-- */
$(0,-1,-1)$ // j:=0; Aux:=0 //
$(0,0,0)$
$(-1,+1,+1)$ // i--; j++; Aux++ //
$(0,0,0)$
$(+1,0,-1)$ // i++; Aux-- //
$(0,0,0)$
$(0,-1,0)$ // j-- //

Figure 1 A skeleton of a simple imperative program (left) and its VASS model (right).

$\mathcal{C}[c] : \mathbb{N} \to \mathbb{N}_\infty$ such that $\mathcal{C}[c](n)$ is the maximal value of c along a computation initiated in a configuration with all counters set to n. So far, three types of VASS models have been investigated in previous works.

- *Demonic VASS*, where the non-determinism is resolved by an adversarial environment aiming to increase the complexity.
- *VASS Games*, where every control state is declared as *angelic* or *demonic*, and the non-determinism is resolved by the controller or by the environment aiming to lower and increase the complexity, respectively.
- *VASS MDPs*, where the states are either non-deterministic or stochastic. The non-determinism is usually resolved in the "demonic" way.

Let us note that the "angelic" and "demonic" non-determinism are standard concepts in program analysis [6] applicable to arbitrary computational devices including VASS. The use of VASS termination/counter complexity analysis is illustrated in the next example.

▶ **Example 1.** Consider the program skeleton of Fig. 1 (left). Since a VASS cannot directly model the assignment `j:=i` and cannot test a counter for zero, the skeleton is first transformed into an equivalent program of Fig. 1 (middle), where the assignment `j:=i` is implemented using an auxiliary variable `Aux` and two `while` loops. Clearly, the execution of the transformed program is only longer than the execution of the original skeleton (for all inputs). For the transformed program, an over-approximating demonic VASS model is obtained by replacing conditionals with non-determinism, see Fig. 1 (right). When all counters are initialized to n, the VASS terminates after $\mathcal{O}(n^2)$ transitions. Hence, the same upper bound is valid also for the original program skeleton. Actually, the run-time complexity of the skeleton is $\Theta(n^2)$ where n is the initial value of `i`, so the obtained upper bound is asymptotically optimal.

Existing results. In [5], it is shown that the problem whether $\mathcal{L} \in \mathcal{O}(n)$ for a given demonic VASS is solvable in polynomial time, and a complete proof method based on linear ranking functions is designed. The polynomiality of termination complexity for a given demonic VASS is also decidable in polynomial time, and if $\mathcal{L} \notin \mathcal{O}(n^k)$ for any $k \in \mathbb{N}$, then $\mathcal{L} \in 2^{\Omega(n)}$ [14]. The same results hold for counter complexity. In [20], a polynomial time algorithm computing the *least* $k \in \mathbb{N}$ such that $\mathcal{L} \in \mathcal{O}(n^k)$ for a given demonic VASS is presented (the algorithm first checks if such a k exists). It is also shown that if $\mathcal{L} \notin \mathcal{O}(n^k)$, then $\mathcal{L} \in \Omega(n^{k+1})$. Again, the same results hold also for counter complexity. The proof is actually given only for *strongly connected demonic VASS*, and it is conjectured that a generalization to unrestricted demonic VASS can be obtained by extending the presented construction (see the Introduction of [20]). In [13], it was shown that the problem whether the termination/counter complexity of a given demonic VASS belongs to a given level of Grzegorczyk hierarchy is solvable in polynomial time, and the same problem for VASS games is shown **NP**-complete. The **NP** upper bound follows by observing that player Angel can safely commit to a *countreless*[1] strategy when minimizing the complexity level in the Grzegorczyk hierarchy. Intuitively, this is because Grzegorczyk classes are closed under function composition (unlike the classes $\Theta(n^k)$). Furthermore, the problem whether $\mathcal{L} \in \mathcal{O}(n^2)$ for a given VASS game is shown **PSPACE** hard, but the decidability of this problem is left open. As for VASS MDPs, the only existing result is [4], where it is shown that the linearity of termination complexity is solvable in polynomial time for VASS MDPs with a tree-like MEC decomposition.

Our contribution. For demonic VASS, we *refute* the conjecture of [20] and prove that for general (not necessarily strongly connected) demonic VASS, the problem whether

- $\mathcal{L} \in \mathcal{O}(n^k)$ is in **P** for $k = 1$, and **coNP**-complete for $k \geq 2$;
- $\mathcal{L} \in \Omega(n^k)$ is in **P** for $k \leq 2$, and **NP**-complete for $k \geq 3$;
- $\mathcal{L} \in \Theta(n^k)$ is in **P** for $k = 1$, **coNP**-complete for $k = 2$, and **DP**-complete for $k \geq 3$.

The same results are proven also for counter complexity.

Since the demonic VASS constructed in our proofs are relatively complicated, we write them in a simple imperative language with a precisely defined VASS semantics. This allows to present the overall proof idea clearly and justify technical correctness by following the control flow of the VASS program, examining possible side effects of the underlying “gadgets”, and verifying that the Demon does not gain anything by deviating from the ideal execution scenario.

When proving the upper bounds, we show that every path in the DAG of strongly connected components can be associated with the (unique) vector describing the maximal simultaneous increase of the counters. Here, the counters pumpable to exponential (or even larger) values require special treatment. We show that this vector is computable in polynomial time. Hence, the complexity of a given counter c is $\Omega(n^k)$ iff there is a path in the DAG such that the associated maximal increase of c is $\Omega(n^k)$. Thus, we obtain the **NP** upper bound, and the other upper bounds follow similarly. The crucial parameter characterizing hard-to-analyze instances is the number of different paths from a root to a leaf in the DAG decomposition, and tractable subclasses of demonic VASS are obtained by bounding this parameter. We refer to Section 3 for more details.

Then, we turn our attention to VASS games, where the problem of polynomial termination/counter complexity analysis requires completely new ideas. In [13], it was observed that the information about the “asymptotic counter increase performed so far” must be taken into account by player Angel when minimizing the complexity level in the polynomial hierarchy, and counterless strategies are therefore insufficient. However, it is not clear what information is needed to make optimal decisions, and whether this information is finitely representable. We show that player Angel can safely commit to a so-called *locking* strategy. A strategy for player Angel is *locking* if whenever a new angelic state p is visited, one of its outgoing transition is chosen and “locked” so that when p is revisited, the same locked transition is used. The locked transition choice may depend on the computational history and the transitions locked in previously visited angelic states. Then, we define a *locking decomposition*

[1] A strategy is *counterless* if the decision depends just on the control state of the configuration currently visited.

```
1 input i;
2 j:=0; k:=0; z:=0;
3 if condition    // demonic choice //
4     then while (i>0) do j++; k:=k+i; i--; done
5     else j:=i*i; k:=i;
6          while (i>0) do j:=j+k; i--; done
7 choose:         // angelic choice //
8     while (j>0) do j--; z++ done
9 or: while (k>0) do k--; z++ done
```

Figure 2 A simple program with both demonic and angelic non-determinism.

of a given VASS that plays a role similar to the DAG decomposition for demonic VASS. Using the locking decomposition, the existence of a suitable locking strategy for player Angel is decided by an alternating polynomial time algorithm (and hence in polynomial space). Thus, we obtain the following: For every VASS game, we have that $\mathcal{L}$ is either in $\mathcal{O}(n^k)$ or in $\Omega(n^{k+1})$. Furthermore, the problem whether

- $\mathcal{L} \in \mathcal{O}(n^k)$ is **NP**-complete for k=1 and **PSPACE**-complete for k≥2;
- $\mathcal{L} \in \Omega(n^k)$ is in **P** for k=1, **coNP**-complete for k=2, and **PSPACE**-complete for k≥3;
- $\mathcal{L} \in \Theta(n^k)$ is **NP**-complete for k=1 and **PSPACE**-complete for k≥2.

The same results hold also for counter complexity. Similarly to demonic VASS, tractable subclasses of VASS games are obtained by bounding the number of different paths in the locking decomposition.

The VASS model constructed in Example 1 is purely demonic. The use of VASS *games* in program analysis/synthesis is illustrated in the next example.

▶ **Example 2.** Consider the program of Fig. 2. The `condition` at line 3 is resolved by the environment in a demonic way. The two branches of `if-then-else` execute a code modifying the variables `j` and `k`. After that, the controller can choose one of the two **while**-loops at lines 8, 9 with the aim of keeping the value of `z` small. The question is how the size of `z` grows with the size of input if the controller makes optimal decisions. A closer look reveals that when the variable `i` is assigned n at line 1, then

- the values of `j` and `k` are $\Theta(n)$ and $\Theta(n^2)$ when the `condition` is evaluated to *true*;
- the values of `j` and `k` are $\Theta(n^2)$ and $\Theta(n)$ when the `condition` is evaluated to *false*.

Hence, the controller can keep `z` in $\Theta(n)$ if an optimal decision is taken. Constructing a VASS game model for the program of Fig. 2 is straightforward (the required gadgets are given in Fig. 3). Using the results of this paper, the above analysis can be performed *fully automatically.*

2 Preliminaries

The sets of integers and non-negative integers are denoted by $\mathbb{Z}$ and $\mathbb{N}$, respectively, and we use $\mathbb{N}_\infty$ to denote $\mathbb{N} \cup \{\infty\}$. The vectors of $\mathbb{Z}^d$ where $d \geq 1$ are denoted by $\mathbf{v}, \mathbf{u}, \ldots$, and the vector $(n, \ldots, n)$ is denoted by $\vec{n}$.

▶ **Definition 3** (VASS). *Let $d \geq 1$. A d-*dimensional vector addition system with states (VASS) *is a pair $\mathcal{A} = (Q, \mathit{Tran})$, where $Q \neq \emptyset$ is a finite set of* states *and $\mathit{Tran} \subseteq Q \times \mathbb{Z}^d \times Q$ is a finite set of* transitions *such that for every $q \in Q$ there exist $p \in Q$ and $\mathbf{u} \in \mathbb{Z}^d$ such that $(q, \mathbf{u}, p) \in \mathit{Tran}$.*

The set Q is split into two disjoint subsets Q_A and Q_D of *angelic* and *demonic* states controlled by the players Angel and Demon, respectively. A *configuration* of $\mathcal{A}$ is a pair $p\mathbf{v} \in Q \times \mathbb{N}^d$, where $\mathbf{v}$ is the vector of counter values. We often refer to counters by their symbolic names. For example, when we say that $\mathcal{A}$ has three counters x, y, z and the value of x in a configuration $p\mathbf{v}$ is 8, we mean that $d = 3$ and $\mathbf{v}_i = 8$ where i is the index associated to x. When the mapping between a counter name and its index is essential, we use c_i to denote the counter with index i.

A *finite path* in $\mathcal{A}$ of length m is a finite sequence $\varrho = p_1, \mathbf{u}_1, p_2, \mathbf{u}_2, \ldots, p_m$ such that $(p_i, \mathbf{u}_i, p_{i+1}) \in \mathit{Tran}$ for all $1 \leq i < m$. We use $\Delta(\varrho)$ to denote the *effect* of ϱ, defined as $\sum_{i=1}^{m} \mathbf{u}_i$. An *infinite path* in $\mathcal{A}$ is an infinite sequence $\alpha = p_1, \mathbf{u}_1, p_2, \mathbf{u}_2, \ldots$ such that every finite prefix $p_1, \mathbf{u}_1, \ldots, p_m$ of α is a finite path in $\mathcal{A}$.

A *computation* of $\mathcal{A}$ is a sequence of configurations $\alpha = p_1\mathbf{v}_1, p_2\mathbf{v}_2, \ldots$ of length $m \in \mathbb{N}_\infty$ such that for every $1 \leq i < m$ there is a transition $(p_i, \mathbf{u}_i, p_{i+1})$ satisfying $\mathbf{v}_{i+1} = \mathbf{v}_i + \mathbf{u}_i$. Note that every computation determines its associated path in the natural way.

VASS Termination Complexity. A *strategy* for Angel (or Demon) in $\mathcal{A}$ is a function η assigning to every finite computation $p_1\mathbf{v}_1, \ldots, p_m\mathbf{v}_m$ where $p_m \in Q_A$ (or $p_m \in Q_D$) a transition $(p_m, \mathbf{u}, q)$. Every pair of strategies (σ, π) for Angel/Demon and every initial configuration $p\mathbf{v}$ determine the unique *maximal* computation $\mathit{Comp}^{\sigma,\pi}(p\mathbf{v})$ initiated in $p\mathbf{v}$. The maximality means that the computation cannot be prolonged without making some counter negative. For a given counter c, we use $\mathit{max}[c](\mathit{Comp}^{\sigma,\pi}(p\mathbf{v}))$ to denote the supremum of the c's values in all configurations visited along $\mathit{Comp}^{\sigma,\pi}(p\mathbf{v})$. Furthermore, we use $\mathit{len}(\mathit{Comp}^{\sigma,\pi}(p\mathbf{v}))$ to denote the length of $\mathit{Comp}^{\sigma,\pi}(p\mathbf{v})$. Note that $\mathit{max}[c]$ and len can be infinite for certain computations.

For every initial configuration $p\mathbf{v}$, consider a game where the players Angel and Demon aim at minimizing and maximizing the $\mathit{max}[c]$ or len objective, respectively. By applying standard game-theoretic arguments (see [1] for an explicit proof), we obtain

$$\sup_{\pi} \inf_{\sigma} \mathit{len}(\mathit{Comp}^{\sigma,\pi}(p\mathbf{v})) = \inf_{\sigma} \sup_{\pi} \mathit{len}(\mathit{Comp}^{\sigma,\pi}(p\mathbf{v})) \tag{1}$$

$$\sup_{\pi} \inf_{\sigma} \mathit{max}[c](\mathit{Comp}^{\sigma,\pi}(p\mathbf{v})) = \inf_{\sigma} \sup_{\pi} \mathit{max}[c](\mathit{Comp}^{\sigma,\pi}(p\mathbf{v})) \tag{2}$$

where σ and π range over all strategies for Angel and Demon, respectively. Hence, there exists a unique *termination value* of $p\mathbf{v}$, denoted by $\mathit{Tval}(p\mathbf{v})$, defined by (1). Similarly, for every counter c there exists a unique *maximal counter value*, denoted by $\mathit{Cval}[c](p\mathbf{v})$, defined by (2). Furthermore, both players have *optimal* positional strategies σ^* and π^* achieving the outcome specified by the equilibrium value or better in every configuration $p\mathbf{v}$ against every strategy of the opponent (here, a *positional* strategy is a strategy depending only on the currently visited configuration). We refer to [1] for details.

The *termination complexity* and *c-counter complexity* of $\mathcal{A}$ are functions $\mathcal{L}, \mathcal{C}[c] : \mathbb{N} \to \mathbb{N}_\infty$ where $\mathcal{L}(n) = \max\{\mathit{Tval}(p\vec{n}) \mid p \in Q\}$ and $\mathcal{C}[c](n) = \max\{\mathit{Cval}[c](p\vec{n}) \mid p \in Q\}$. When the underlying VASS $\mathcal{A}$ is not clear, we write $\mathcal{L}_\mathcal{A}$ and $\mathcal{C}_\mathcal{A}[c]$ instead of $\mathcal{L}$ and $\mathcal{C}[c]$.

Observe that the asymptotic analysis of termination complexity for a given VASS $\mathcal{A}$ is trivially reducible to the asymptotic analysis of counter complexity in a VASS $\mathcal{B}$ obtained from $\mathcal{A}$ by adding a fresh "step counter" sc incremented by every transition of $\mathcal{B}$. Clearly, $\mathcal{L}_\mathcal{A} \in \Theta(\mathcal{C}_\mathcal{B}[sc])$. Therefore, the lower complexity bounds for the considered problems of asymptotic analysis are proven for $\mathcal{L}$, while the upper bounds are proven for $\mathcal{C}[c]$.

VASS Program 1 $\mathcal{A}_\varphi$.

```
$d_2$ += $d_1 * e_1$;  $d_3$ += $d_2 * e_2$;  $\cdots$ ;  $d_k$ += $d_{k-1} * e_{k-1}$;
foreach $i = 1, \ldots, v$ do
    choose: $x_i$ += $d_k$ or $\bar{x}_i$ += $d_k$;
end
$s_0$ += $d_k$;
foreach $i = 1, \ldots, m$ do
    choose: $s_i$ += $\min(\ell_1^i, s_{i-1})$ or $s_i$ += $\min(\ell_2^i, s_{i-1})$ or $s_i$ += $\min(\ell_3^i, s_{i-1})$;
end
$f$ += $s_m * n$
```

3 Demonic VASS

In this section, we classify the computational complexity of polynomial asymptotic analysis for demonic VASS. The following theorem holds regardless whether the counter update vectors are encoded in unary or binary (the lower bounds hold for unary encoding, the upper bounds hold for binary encoding).

▶ **Theorem 4.** *Let $k \geq 1$. For every demonic VASS $\mathcal{A}$ we have that $\mathcal{L}$ is either in $\mathcal{O}(n^k)$ or in $\Omega(n^{k+1})$. Furthermore, the problem whether*

- $\mathcal{L} \in \mathcal{O}(n^k)$ *is in* **P** *for $k = 1$, and* **coNP**-*complete for $k \geq 2$;*
- $\mathcal{L} \in \Omega(n^k)$ *is in* **P** *for $k \leq 2$, and* **NP**-*complete for $k \geq 3$;*
- $\mathcal{L} \in \Theta(n^k)$ *is in* **P** *for $k = 1$,* **coNP**-*complete for $k = 2$, and* **DP**-*complete for $k \geq 3$.*

The same results hold also for $\mathcal{C}[c]$ (for a given counter c of $\mathcal{A}$).

The next theorem identifies the crucial parameter influencing the complexity of polynomial asymptotic analysis for demonic VASS. Let $\mathcal{D}(\mathcal{A})$ be the standard DAG of strongly connected components of $\mathcal{A}$. For every leaf (bottom SCC) η of $\mathcal{D}(\mathcal{A})$, let $Deg(\eta)$ be the total number of all paths from a root of $\mathcal{D}(\mathcal{A})$ to η.

▶ **Theorem 5.** *Let Λ be a class of demonic VASS such that for every $\mathcal{A} \in \Lambda$ and every leaf η of $\mathcal{D}(\mathcal{A})$ we have that $Deg(\eta)$ is bounded by a fixed constant depending only on Λ.*

Then, the problems whether $\mathcal{L}_\mathcal{A} \in \mathcal{O}(n^k)$, $\mathcal{L}_\mathcal{A} \in \Omega(n^k)$, $\mathcal{L}_\mathcal{A} \in \Theta(n^k)$ for given $\mathcal{A} \in \Lambda$ and $k \in \mathbb{N}$, are solvable in polynomial time (where the k is written in binary). The same results hold also for $\mathcal{C}[c]$ (for a given counter c of $\mathcal{A}$).

The degree of the polynomial bounding the running time of the decision algorithm for the three problems of Theorem 5 increases with the increasing size of the constant bounding $Deg(\eta)$. From the point of view of program analysis, Theorem 5 has a clear intuitive meaning. If $\mathcal{A}$ is an abstraction of a program $\mathcal{P}$, then the instructions in $\mathcal{P}$ increasing the complexity of the asymptotic analysis of $\mathcal{A}$ are *branching instructions* such as **if-then-else** that are *not embedded within loops*. If $\mathcal{P}$ executes many such constructs in a sequence, a termination point can be reached in many ways ("zigzags" in the $\mathcal{P}$'s control-flow graph). This increases $Deg(\eta)$, where η is a leaf of $\mathcal{D}(\mathcal{A})$ containing the control state modeling the termination point of $\mathcal{P}$.

3.1 Lower bounds

Since the asymptotic analysis of $\mathcal{L}$ is trivially reducible to the asymptotic analysis of $\mathcal{C}[c]$ (see Section 2), all lower complexity bounds of Theorem 4 follow directly from the next two lemmata.

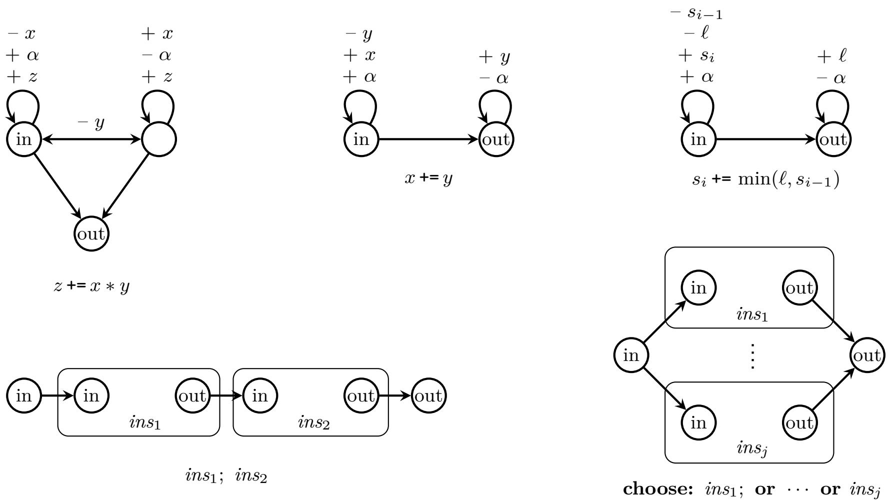

$-x$
$+\alpha$
$+z$
$+x$
$-\alpha$
$+z$
$-y$
in
out
$z \mathrel{+}= x * y$
$-y$
$+x$
$+\alpha$
$+y$
$-\alpha$
in
out
$x \mathrel{+}= y$
$-s_{i-1}$
$-\ell$
$+s_i$
$+\alpha$
$+\ell$
$-\alpha$
in
out
$s_i \mathrel{+}= \min(\ell, s_{i-1})$
in
in
out
ins_1
in
out
ins_2
out
$ins_1;\ ins_2$
in
out
ins_1
in
out
ins_j
in
out
choose: ins_1; **or** $\cdots$ **or** ins_j

Figure 3 The gadgets of $\mathcal{A}_\varphi$.

▶ **Lemma 6.** *Let $k \geq 2$. For every propositional formula φ in 3-CNF there exists a demonic VASS $\mathcal{A}_\varphi$ constructible in time polynomial in $|\varphi|$ such that*

- *if φ is satisfiable, then $\mathcal{L}_{\mathcal{A}_\varphi} \in \Theta(n^{k+1})$;*
- *if φ is not satisfiable, then $\mathcal{L}_{\mathcal{A}_\varphi} \in \Theta(n^k)$.*

Proof. Let $\varphi \equiv C_1 \wedge \cdots \wedge C_m$ be a propositional formula where every $C_i \equiv \ell_1^i \vee \ell_2^i \vee \ell_3^i$ is a clause with three literals over propositional variables $X_1, \ldots, X_v$ (a literal is a propositional variable or its negation). We construct a VASS $\mathcal{A}_\varphi$ with the counters

- $x_1, \cdots, x_v, \bar{x}_1, \cdots, \bar{x}_v$ used to encode an assignment of truth values to $X_1, \ldots, X_v$. In the following, we identify literals ℓ_j^i of φ with their corresponding counters (i.e., if $\ell_j^i \equiv X_u$, the corresponding counter is x_u; and if $\ell_j^i \equiv \neg X_u$, the corresponding counter is $\bar{x}_u$).
- $s_0, \ldots, s_m$ used to encode the validity of clauses under the chosen assignment,
- f used to encode the (in)validity of φ under the chosen assignment,
- $d_1, \ldots, d_k$ and $e_1, \ldots, e_{k-1}$ used to compute n^k,
- and some auxiliary counters used in gadgets.

The structure of $\mathcal{A}_\varphi$ is shown in VASS Program 1. The basic instructions are implemented by the gadgets of Fig. 3 (top). Counter changes associated to a given transition are indicated by the corresponding labels, where $-c$ and $+c$ mean decrementing and incrementing a given counter by one (the other counters are unchanged). Hence, the empty label represents no counter change, i.e., the associated counter update vector is $\vec{0}$. The auxiliary counter α is *unique for every instance* of these gadgets and it is not modified anywhere else.

The constructs ins_1; ins_2 and **choose:** ins_1; **or** $\cdots$ **or** ins_j are implemented by connecting the underlying gadgets as shown in Fig. 3 (bottom). The **foreach** statements are just concise representations of the corresponding sequences of instructions connected by ';'.

Now suppose that the computation of VASS Program 1 is executed from line 1 where all counters are initialized to n. One can easily verify that all gadgets implement the operations associated to their labels up to some "asymptotically irrelevant side effects". More precisely,

- the $z \mathrel{+}= x * y$ gadget ensures that the Demon can increase the value of counter z by $val(x) + val(y) \cdot (val(x) + n)$ (but not more) if he plays optimally, where $val(x)$ and $val(y)$ are the values stored in x and y when initiating the gadget. Recall that the counter α is unique for the gadget, and its initial value is n. Also note that the value of y is decreased to 0 when the Demon strives to maximally increase the value of z.

- The x += y gadget ensures that the Demon can add $val(y)$ to the counter x and then reset y to the value $val(y) + n$ (but not more) if he plays optimally. Again, note that α is a unique counter for the gadget with initial value n.
- The s_i += $\min(\ell, s_{i-1})$ gadget allows the Demon to increase s_i by the minimum of $val(\ell)$ and $val(s_{i-1})$, and then restore ℓ to $val(\ell) + n$ (but not more).

Now, the VASS Program 1 is easy to follow. We describe its execution under the assumption that the Demon plays *optimally.* It is easy to verify that the Demon cannot gain anything by deviating from the below described scenario where certain counters are pumped to their *maximal* values (in particular, the auxiliary counters are never re-used outside their gadgets, hence the Demon is not motivated to leave any positive values in them).

By executing line 1, the Demon pumps the counter d_k to the value $\Theta(n^k)$. Then, the Demon determines a truth assignment for every X_i, where $i \in \{1, \ldots, v\}$, by pumping either the counter x_i or the counter $\bar{x}_i$ to the value $\Theta(n^k)$. A key observation is that when the chosen assignment makes φ true, then every clause contains a literal such that the value of its associated counter is $\Theta(n^k)$. Otherwise, there is a clause C_i such that all of the three counters corresponding to ℓ_1^i, ℓ_2^i, ℓ_3^i have the value n. The Demon continues by pumping s_0 to the value $\Theta(n^k)$ at line 5. Then, for every $i = 1, \ldots, m$, he selects a literal ℓ_j^i of C_i and pumps s_i to the minimum of $val(s_{i-1})$ and $val(\ell_j^i)$. Observe that $val(s_{i-1})$ is either $\Theta(n)$ or $\Theta(n^k)$, and the same holds for $val(s_i)$ after executing the instruction. Hence, s_m is pumped either to $\Theta(n^k)$ or $\Theta(n)$, depending on whether the chosen assignment sets every clause to true or not, respectively. Note that the length of the whole computation up to line 9 is $\Theta(n^k)$, regardless whether the chosen assignment sets the formula φ to true or false. If s_m was pumped to $\Theta(n^k)$, then the last instruction at line 9 can pump the counter f to $\Theta(n^{k+1})$ in $\Theta(n^{k+1})$ transitions. Hence, if φ is satisfiable, the Demon can schedule a computation of length $\Theta(n^{k+1})$. Otherwise, the length of the longest computation is $\Theta(n^k)$. Also observe that if the Demon starts executing $\mathcal{A}_\varphi$ in some other control state (i.e., not in the first instruction of line 1), the maximal length of a computation is only shorter. ◀

Recall that the class **DP** consists of problems that are intersections of one problem in **NP** and another problem in **coNP**. The class **DP** is expected to be somewhat larger than **NP** ∪ **coNP**, and it is contained in the $\mathbf{P}^{\mathbf{NP}}$ level of the polynomial hierarchy. The standard **DP**-complete problem is SAT-UNSAT, where an instance is a pair φ, ψ of propositional formulae and the question is whether φ is satisfiable and ψ is unsatisfiable [17]. Hence, the **DP** lower bounds of Theorem 4 follow directly from the next lemma (a proof can be found in [1]).

▶ **Lemma 7.** *Let $k \geq 3$. For every pair φ, ψ of propositional formulae in 3-CNF there exists a demonic VASS $\mathcal{A}_{\varphi,\psi}$ such that $\mathcal{L}_{\mathcal{A}_{\varphi,\psi}} \in \Theta(n^k)$ iff φ is satisfiable and ψ is unsatisfiable.*

3.2 Upper bounds

The upper complexity bounds of Theorem 4 are proven for $\mathcal{C}[c]$. For the sake of clarity, we first sketch the main idea and then continue with developing a formal proof.

Intuition. For a given demonic VASS $\mathcal{A}$, we compute its SCC decomposition and proceed by analyzing the individual SCCs in the way indicated in Fig. 4. We start in a top SCC with all counters initialized to n. Here, we can directly apply the results of [20, 14] and decide in polynomial time whether $\mathcal{C}[c] \in \Theta(n^k)$ for some $k \in \mathbb{N}$ or $\mathcal{C}[c] \in 2^{\Omega(n)}$ (in the first case, we can also determine the k). We perform this analysis for every counter c and thus obtain the vector describing the maximal asymptotic growth of the counters (such as $(n^2, n, 2^{\Omega(n)})$ in Fig. 4). Observe that

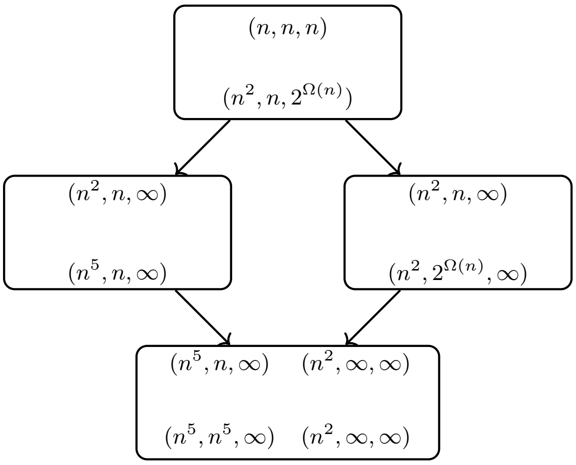


Figure 4 Analyzing $\mathcal{C}[c]$ in a demonic VASS by SCC decomposition.

- although the asymptotic growth of $\mathcal{C}[c]$ has been analyzed for each counter independently, all counters can be pumped to their associated asymptotic values *simultaneously*. Intuitively, this is achieved by considering the "pumping computations" for the smaller vector $(\lfloor n/d \rfloor, \ldots, \lfloor n/d \rfloor)$ of initial counter values (d is the dimension of $\mathcal{A}$), and then simply "concatenating" these computations in a configuration with all counters initialized to n;
- if the asymptotic growth of $\mathcal{C}[c]$ is $\Theta(n)$, the computation simultaneously pumping the counters to their asymptotic values may actually *decrease* the value of c (the computation can be arranged so that the resulting value of c stays above $\lfloor n/d \rfloor$ for all sufficiently large n). For example, the top SCC of Fig. 4 achieves the simultaneous asymptotic growth of all counters from (n,n,n) to $(n^2,n,2^{\Omega(n)})$, but this does *not* imply the counters can be simultaneously increased above the original value n (nevertheless, the simultaneous increase in the first and the third counter above n is certainly possible for all sufficiently large n).

A natural idea how to proceed with next SCCs is to perform a similar analysis for larger vectors of initial counter values. Since we are interested just in the asymptotic growth of the counters, we can safely set the initial value of a counter previously pumped to $\Theta(n^k)$ to *precisely* n^k. However, it is not immediately clear how to treat the counters previously pumped to $2^{\Omega(n)}$. We show that the length of a computation "pumping" the counters to their new asymptotic values in the considered SCC $\mathcal{C}$ is at most exponential in n. Consequently, the "pumping computation" in $\mathcal{C}$ can be constructed so that the resulting value of the "large" counters stays above one half of their original value. This means the value of "large" counters is still in $2^{\Omega(n)}$ after completing the computation in $\mathcal{C}$. Furthermore, the large counters can be treated as if their initial value was infinite when analyzing $\mathcal{C}$. This "infinite" initial value is implemented simply by modifying every counter update vector $\mathbf{u}$ in $\mathcal{C}$ so that $\mathbf{u}[j]=0$ for every "large" counter c_j. This adjustment in the structure of $\mathcal{C}$ is denoted by putting "∞" into the corresponding component of the initial counter value vector (see Fig. 4). This procedure is continued until processing all SCCs. Note that the same SCC may be processed *multiple times* for different vectors of initial counter values corresponding to different paths from a top SCC. In Fig. 4, the bottom SCC is processed for the initial vectors (n^5,n,∞) and (n^2,∞,∞) corresponding to the two paths from the top SCC. The number of such initial vectors can be *exponential* in the size of $\mathcal{A}$, as witnessed by the VASS constructed in the proof of Lemma 6.

Now we give a formal proof. Let $\mathcal{A}$ be a demonic VASS with d counters. For every counter c and every $\mathbf{v} \in \mathbb{N}^d$, we define the function $\mathcal{C}[c,\mathbf{v}] : \mathbb{N} \to \mathbb{N}_\infty$ where $\mathcal{C}[c,\mathbf{v}](n)$ is the maximum of all $\mathit{Cval}[c](p\mathbf{u})$ where $p \in Q$ and $\mathbf{u} = (n^{\mathbf{v}(1)}, \ldots, n^{\mathbf{v}(d)})$.

▶ **Proposition 8.** *Let $\mathcal{A}$ be a strongly connected demonic VASS with d counters, and let $\mathbf{v} \in \mathbb{N}^d$ such that $\mathbf{v}(i) \leq 2^{j \cdot d}$ for every $i \leq d$, where $j < |Q|$. For every counter c, we have that either $\mathcal{C}[c,\mathbf{v}] \in \Theta(n^k)$ for some $1 \leq k \leq 2^{(j+1)\cdot d}$, or $\mathcal{C}[c,\mathbf{v}] \in \Omega(2^n)$. It is decidable in polynomial time which of the two possibilities holds. In the first case, the value of k is computable in polynomial time.*

In [20], a special variant of Proposition 8 covering the subcase when $\mathbf{v} = \vec{1}$ is proven. In the introduction part of [20], it is mentioned that a generalization of this result (equivalent to Proposition 8) can be obtained by modifying the techniques presented in [20]. Although no explicit proof is given, the modification appears feasible. We give a simple explicit proof of Proposition 8, using the algorithm of [20] for the $\mathbf{v} = \vec{1}$ subcase as a “black-box procedure”. We refer to [1] for details.

Now we extend the function $\mathcal{C}[c,\mathbf{v}]$ so that $\mathbf{v} \in \mathbb{N}_\infty^d$. Here, the ∞ components of $\mathbf{v}$ correspond to counters that have already been pumped to “very large” values and do not constrain the computations in $\mathcal{A}$. As we shall see, “very large” actually means “at least singly exponential in n”.

Let $\mathbf{v} \in \mathbb{N}_\infty^d$, and let $\mathcal{A}_\mathbf{v}$ be the VASS obtained from $\mathcal{A}$ by modifying every counter update vector $\mathbf{u}$ into $\mathbf{u}'$, where $\mathbf{u}'(i) = \mathbf{u}(i)$ if $\mathbf{v}(i) \neq \infty$, otherwise $\mathbf{u}'(i) = 0$. Hence, the counters set to ∞ in $\mathbf{v}$ are never changed in $\mathcal{A}_\mathbf{v}$. Furthermore, let $\mathbf{v}'$ be the vector obtained from $\mathbf{v}$ by changing all ∞ components into 1. We put $\mathcal{C}_\mathcal{A}[c,\mathbf{v}] = \mathcal{C}_{\mathcal{A}_\mathbf{v}}[c,\mathbf{v}']$.

For a given $\mathbf{v} \in \mathbb{N}_\infty^d$, we say that $F : \mathbb{N} \to \mathbb{N}^d$ is $\mathbf{v}$-*consistent* if for every $i \in \{1,\ldots,d\}$ we have that the projection $F_i : \mathbb{N} \to \mathbb{N}$ is either $\Theta(n^k)$ if $\mathbf{v}_i = k$, or $2^{\Omega(n)}$ if $\mathbf{v}_i = \infty$. Intuitively, a $\mathbf{v}$-consistent function assigns to every $n \in \mathbb{N}$ a vector $F(n)$ of initial counter values growing consistently with $\mathbf{v}$.

Given $\mathbf{v} \in \mathbb{N}_\infty^d$, a control state $p \in Q$, a $\mathbf{v}$-consistent function F, an infinite family $\Pi = \pi_1, \pi_2, \ldots$ of Demon's strategies in $\mathcal{A}$, a function $S : \mathbb{N} \to \mathbb{N}$, and $n \in \mathbb{N}$, we use $\beta_n[\mathbf{v}, p, F, \Pi, S]$ to denote the computation of $\mathcal{A}$ starting at $pF(n)$ obtained by applying π_n until a maximal computation is produced or $S(n)$ transitions are executed.

The next lemma says that if $\mathcal{A}$ is strongly connected, then all counters can be pumped *simultaneously* to the values asymptotically equivalent to $\mathcal{C}_\mathcal{A}[c,\mathbf{v}]$ so that the counters previously pumped to exponential values stay exponential.

▶ **Lemma 9.** *Let $\mathcal{A}$ be a strongly connected demonic VASS with d counters. Let $\mathbf{v} \in \mathbb{N}_\infty^d$, and let F be a $\mathbf{v}$-consistent function. Then for every counter c_i such that $\mathbf{v}_i \neq \infty$ and $\mathcal{C}_\mathcal{A}[c_i,\mathbf{v}] \in \Theta(n^k)$ we have that $\mathit{Cval}[c_i](pF(n)) \in \Theta(n^k)$ for every $p \in Q$. Furthermore, there exist $p \in Q$, an infinite family Π of Demon's strategies, and a function $S \in 2^{\mathcal{O}(n)}$ such that for every c_i, the value of c_i in the last configuration of $\beta_n[\mathbf{v}, p, F, \Pi, S]$ is*

- *$\Theta(n^k)$ if $\mathcal{C}_\mathcal{A}[c_i,\mathbf{v}] \in \Theta(n^k)$;*
- *$2^{\Omega(n)}$ if $\mathbf{v}_i = \infty$ or $\mathcal{C}_\mathcal{A}[c_i,\mathbf{v}] \in 2^{\Omega(n)}$.*

A proof of Lemma 9 uses the result of [14] saying that counters pumpable to exponential values can be simultaneously pumped by a computation of exponential length from a configuration where all counters are set to n (the same holds for polynomially bounded counters, where the length of the computation can be bounded even by a polynomial). Using the construction of Proposition 8, these results are extended to our setting with $\mathbf{v}$-consistent initial counter values. Then, the initial counter values are virtually “split into d boxes” of size $\lfloor F(n)/d \rfloor$. The computations pumping the individual counters are then run "each in its own box" for these smaller initial vectors and concatenated. As the computation of one "box" cannot affect any other "box", no computation can undo the effects of previous computations. The details can be found in [1].

Let $\mathcal{V}_{\mathcal{A}} : \mathbb{N}^d_\infty \to \mathbb{N}^d_\infty$ be a function such that, for every $\mathbf{v} \in \mathbb{N}^d_\infty$,

$$\mathcal{V}_{\mathcal{A}}(\mathbf{v})(i) = \begin{cases} k & \text{if } \mathbf{v}_i \neq \infty \text{ and } \mathcal{C}_{\mathcal{A}}[c_i, \mathbf{v}] \in \Theta(n^k), \\ \infty & \text{otherwise.} \end{cases}$$

Note that every SCC (vertex) η of $\mathcal{D}(\mathcal{A})$ can be seen as a strongly connected demonic VASS after deleting all transitions leading from/to the states outside η. If the counters are simultaneously pumped to $\mathbf{v}$-consistent values before entering η, then η can further pump the counters to $\mathcal{V}_\eta(\mathbf{v})$-consistent values (see Lemma 9). According to Lemma 8, $\mathcal{V}_\eta(\mathbf{v})$ is computable in polynomial time for every $\mathbf{v} \in \mathbb{N}^d_\infty$ where every *finite* $\mathbf{v}_i$ is bounded by $2^{j \cdot d}$ for some $j < |Q|$.

Observe that *all* computations of $\mathcal{A}$ can be divided into finitely many pairwise disjoint classes according to their corresponding paths in $\mathcal{D}_{\mathcal{A}}$ (i.e., the sequence of visited SCCs of $\mathcal{D}_{\mathcal{A}}$). For each such sequence $\eta_1, \ldots, \eta_m$, the vectors $\mathbf{v}_0, \ldots, \mathbf{v}_m$ where $\mathbf{v}_0 = \vec{1}$ and $\mathbf{v}_i = \mathcal{V}_{\eta_i}(\mathbf{v}_{i-1})$ are computable in time polynomial in $|\mathcal{A}|$ (note that $m \leq |Q|$). The asymptotic growth of the counters achievable by computations following the path $\eta_1, \ldots, \eta_m$ is then given by $\mathbf{v}_m$. Hence, $\mathcal{C}_{\mathcal{A}}[c_i] \in \Omega(n^k)$ iff there is a path $\eta_1, \ldots, \eta_m$ in $\mathcal{D}_{\mathcal{A}}$ such that $\mathbf{v}_m(i) \geq k$. Similarly, $\mathcal{C}_{\mathcal{A}}[c_i] \in \mathcal{O}(n^k)$ iff for every path $\eta_1, \ldots, \eta_m$ in $\mathcal{D}_{\mathcal{A}}$ we have that $\mathbf{v}_m(i) \leq k$. From this we immediately obtain the upper complexity bounds of Theorem 4.

Furthermore, for every SCC η of $\mathcal{D}_{\mathcal{A}}$, we can compute the set $\mathit{Vectors}_{\mathcal{A}}(\eta)$ of *all* $\mathbf{u}$ such that there is a path $\eta_1, \ldots, \eta_m$ where η_1 is a root of $\mathcal{D}_{\mathcal{A}}$, $\eta_m = \eta$, and $\mathbf{u} = \mathbf{v}_m$. A full description of the algorithm is given in [1]. If $\mathit{Deg}(\eta)$ is bounded by a fixed constant independent of $\mathcal{A}$ for every leaf η of $\mathcal{D}_{\mathcal{A}}$, then the algorithm terminates in polynomial time, which proves Theorem 5.

4 VASS Games

The computational complexity of polynomial asymptotic analysis for VASS games is classified in our next theorem. The parameter characterizing hard instances is identified at the end of this section.

▶ **Theorem 10.** *Let $k \geq 1$. For every VASS game $\mathcal{A}$ we have that $\mathcal{L}$ is either in $\mathcal{O}(n^k)$ or in $\Omega(n^{k+1})$. Furthermore, the problem whether*

- *$\mathcal{L} \in \mathcal{O}(n^k)$ is* **NP***-complete for k=1 and* **PSPACE***-complete for k≥2;*
- *$\mathcal{L} \in \Omega(n^k)$ is in* **P** *for k=1,* **coNP***-complete for k=2, and* **PSPACE***-complete for k≥3;*
- *$\mathcal{L} \in \Theta(n^k)$ is* **NP***-complete for k=1 and* **PSPACE***-complete for k≥2.*

The same results hold also for $\mathcal{C}[c]$ (for a given counter c of $\mathcal{A}$).

Furthermore, we show that for every VASS game $\mathcal{A}$, either $\mathcal{L} \in \mathcal{O}(n^{2^{d|Q|}})$ or $\mathcal{L} \in 2^{\Omega(n)}$. In the first case, the k such that $\mathcal{L} \in \Theta(n^k)$ can be computed in polynomial space. The same results hold for $\mathcal{C}[c]$.

In [13], it has been shown that the problem whether $\mathcal{L} \in \mathcal{O}(n)$ is **NP**-complete, and if $\mathcal{L} \notin \mathcal{O}(n)$, then $\mathcal{L} \in \Omega(n^2)$. This yields the **NP** and **coNP** bounds of Theorem 10 for $k = 1, 2$. Furthermore, it has been shown that the problem whether $\mathcal{L} \in \mathcal{O}(n^2)$ is **PSPACE**-hard, and this proof can be trivially generalized to obtain all **PSPACE** lower bounds of Theorem 10. The details are given in [1].

The key insight behind the proof of Theorem 10 is that player Angel can safely commit to a *simple locking* strategy when minimizing the counter complexity. We start by introducing locking strategies.

▶ **Definition 11.** *Let $\mathcal{A}$ be a VASS game. We say that a strategy σ for player Angel is* locking *if for every computation $p_1\mathbf{v}_1, \ldots, p_m\mathbf{v}_m$ where $p_m \in Q_A$ and for every $k < m$ such that $p_k = p_m$ we have that $\sigma(p_1\mathbf{v}_1, \ldots, p_k\mathbf{v}_k) = \sigma(p_1\mathbf{v}_1, \ldots, p_m\mathbf{v}_m)$.*

In other words, when an angelic control state p is visited for the first time, a locking strategy selects and "locks" an outgoing transition of p so that whenever p is revisited, the previously locked transition is taken. Observe that the choice of a "locked" transition may depend on the whole history of a computation.

Since a "locked" control state has only one outgoing transition, it can be seen as *demonic*. Hence, as more and more control states are locked along a computation, the VASS game $\mathcal{A}$ becomes "more and more demonic". We capture these changes as a finite acyclic graph $\mathcal{G}_{\mathcal{A}}$ called *the locking decomposition of* $\mathcal{A}$. Then, we say that a locking strategy is *simple* if the choice of a locked transition after performing a given history depends only on the finite path in $\mathcal{G}_{\mathcal{A}}$ associated to the history. We show that Angel can achieve an asymptotically optimal termination/counter complexity by using only simple locking strategies. Since the height of $\mathcal{G}_{\mathcal{A}}$ is polynomial in $|\mathcal{A}|$, the existence of an appropriate simple locking strategy for Angel can be decided by an alternating polynomial-time algorithm. As $\mathbf{AP} = \mathbf{PSPACE}$, this proves the **PSPACE** upper bounds of Theorem 10. Furthermore, our construction identifies the structural parameters of $\mathcal{G}_{\mathcal{A}}$ making the polynomial asymptotic analysis of VASS games hard. When these parameters are bounded by fixed constants, the problems of Theorem 10 are solvable in polynomial time.

4.1 Locking sets and the locking decomposition of $\mathcal{A}$

Let $\mathcal{A}$ be a VASS game. A *Demonic decomposion* of $\mathcal{A}$ is a finite directed graph $\mathcal{D}_{\mathcal{A}}$ defined as follows. Let $\sim \subseteq Q \times Q$ be an equivalence where $p \sim q$ iff either $p = q$, or both p, q are demonic and mutually reachable from each other via a finite path leading only through demonic control states. The vertices of $\mathcal{D}_{\mathcal{A}}$ are the equivalence classes $Q/{\sim}$, and $[p] \to [q]$ iff $[p] \neq [q]$ and $(p, \mathbf{u}, q) \in \mathit{Tran}$ for some $\mathbf{u}$. For demonic VASS, $\mathcal{D}_{\mathcal{A}}$ becomes the standard DAG decomposition. For VASS games, $\mathcal{D}_{\mathcal{A}}$ is *not* necessarily acyclic.

A *locking set* of $\mathcal{A}$ is a set of transitions $L \subseteq \mathit{Tran}$ such that $(p, \mathbf{u}, q) \in L$ implies $p \in Q_A$, and $(p, \mathbf{u}, q), (p', \mathbf{u}', q') \in L$ implies $p \neq p'$. A control state p is *locked* by L if L contains an outgoing transition of p. We use $\mathscr{L}$ to denote the set of all locking sets of $\mathcal{A}$. For every $L \in \mathscr{L}$, let $\mathcal{A}_L$ be the VASS game obtained from $\mathcal{A}$ by "locking" the transitions of L. That is, each control state p locked by L becomes demonic in $\mathcal{A}_L$, and the only outgoing transition of p in $\mathcal{A}_L$ is the transition $(p, \mathbf{u}, q) \in L$.

▶ **Definition 12.** *The* locking decomposition of $\mathcal{A}$ *is a finite directed graph $\mathcal{G}_{\mathcal{A}}$ where the set of vertices and the set of edges of $\mathcal{G}_{\mathcal{A}}$ are the least sets V and $\to$ satisfying the following conditions:*

- *All elements of V are pairs $([p], L)$ where $L \in \mathscr{L}$ and $[p]$ is a vertex of $\mathcal{D}_{\mathcal{A}_L}$. When p is demonic/angelic in $\mathcal{A}_L$, we say that $([p], L)$ is demonic/angelic.*
- *V contains all pairs of the form $([p], \emptyset)$.*
- *If $([p], L) \in V$ where p is demonic in $\mathcal{A}_L$ and $[p] \to [q]$ is an edge of $\mathcal{D}_{\mathcal{A}_L}$, then $([q], L) \in V$ and $([p], L) \to ([q], L)$.*
- *If $([p], L) \in V$ where p is angelic in $\mathcal{A}_L$, then for every $(p, \mathbf{u}, q) \in \mathit{Tran}$ we have that $([q], L') \in V$ and $([p], L) \to ([q], L')$, where $L' = L \cup \{(p, \mathbf{u}, q)\}$.*

It is easy to see that $\mathcal{G}_\mathcal{A}$ is *acyclic* and the length of every path in $\mathcal{G}_\mathcal{A}$ is bounded by $|Q| + |Q_A|$, where at most $|Q|$ vertices in the path are demonic. Note that every computation of $\mathcal{A}$ obtained by applying a locking strategy determines its associated path in $\mathcal{G}_\mathcal{A}$ in the natural way. A locking strategy σ is *simple* if the choice of a locked transition depends only on the path in $\mathcal{G}_\mathcal{A}$ associated to the performed history. (i.e., if two histories determine the same path in $\mathcal{G}_\mathcal{A}$, then the strategy makes the same choice for both histories).

4.2 Upper bounds

Let $\mathcal{A}$ be a VASS game with d counters. For every $p \in Q$ and $\mathbf{v} \in \mathbb{N}^d$, let $\mathcal{C}^p_\mathcal{A}[c,\mathbf{v}](n) = \mathit{Cval}[c](p\mathbf{u})$ where $\mathbf{u} = (n^{\mathbf{v}(1)}, \ldots, n^{\mathbf{v}(d)})$. We extend this notation to the vectors $\mathbf{v} \in \mathbb{N}^d_\infty$ in the same way as in Section 3.2, i.e., for a given $\mathbf{v} \in \mathbb{N}^d_\infty$, we put $\mathcal{C}^p_\mathcal{A}[c,\mathbf{v}] = \mathcal{C}^p_{\mathcal{A}_\mathbf{v}}[c,\mathbf{v}']$. Recall that $\mathbf{v}'$ is the vector obtained from $\mathbf{v}$ by changing all ∞ components into 1, and $\mathcal{A}_\mathbf{v}$ is the VASS obtained from $\mathcal{A}$ by modifying every counter update vector $\mathbf{u}$ into $\mathbf{u}'$, where $\mathbf{u}'(i) = \mathbf{u}(i)$ if $\mathbf{v}(i) \neq \infty$, otherwise $\mathbf{u}'(i) = 0$. The main technical step towards obtaining the **PSPACE** upper bounds of Theorem 10 is the next proposition.

▶ **Proposition 13.** *Let $\mathcal{A}$ be a VASS game with d counters. Furthermore, let $([p], L)$ be a vertex of $\mathcal{G}_\mathcal{A}$, $\mathbf{v} \in \mathbb{N}^d_\infty$, and c_i a counter such that $\mathbf{v}_i \neq \infty$. Then, one of the following two possibilities holds:*

- *there is $k \in \mathbb{N}$ such that for every $\mathbf{v}$-consistent F there exist a simple locking Angel's strategy $\sigma_\mathbf{v}$ in $\mathcal{A}_L$ and a Demon's strategy $\pi_\mathbf{v}$ in $\mathcal{A}_L$ such that $\sigma_\mathbf{v}$ is independent of F and*
  - *for every Demon's strategy π in $\mathcal{A}_L$, we have that $\mathit{max}[c_i](\mathit{Comp}^{\sigma_\mathbf{v},\pi}_{\mathcal{A}_L}(p\,F(n))) \in \mathcal{O}(n^k)$;*
  - *for every Angel's strategy σ in $\mathcal{A}_L$, we have that $\mathit{max}[c_i](\mathit{Comp}^{\sigma,\pi_\mathbf{v}}_{\mathcal{A}_L}(p\,F(n))) \in \Omega(n^k)$.*
- *for every $\mathbf{v}$-consistent F there is a Demon's strategy $\pi_\mathbf{v}$ in $\mathcal{A}_L$ such that for every Angel's strategy σ in $\mathcal{A}_L$, we have that $\mathit{max}[c_i](\mathit{Comp}^{\sigma,\pi_\mathbf{v}}_{\mathcal{A}_L}(p\,F(n))) \in 2^{\Omega(n)}$.*

Proposition 13 is proven by induction on the height of the subgraph rooted by $([p], L)$. The case when $([p], L)$ is demonic (which includes the base case when $([p], L)$ is a leaf) follows from the constructions used in the proof of Proposition 8. When the vertex $([p], L)$ is angelic, it has immediate successors of the form $([q_i], L_i)$ where $L_i = L \cup \{(p, \mathbf{u}_i, q_i)\}$. We show that by locking one of the $(p, \mathbf{u}_i, q_i)$ transitions in p, Angel can minimize the growth of c_i in asymptotically the same way as if he used all of these transitions freely when revisiting p. We refer to [1] for details.

Observe that every computation in $\mathcal{A}$ where Angel uses some simple locking strategy determines the unique corresponding path in $\mathcal{G}_\mathcal{A}$ (initiated in a vertex of the form $([p], \emptyset)$) in the natural way. Hence, all such computations can be divided into finitely many pairwise disjoint classes according to their corresponding paths in $\mathcal{G}_\mathcal{A}$. Let $([p_1], L_1), \ldots, ([p_k], L_k)$ be a path in $\mathcal{G}_\mathcal{A}$ where $L_1 = \emptyset$. Consider the corresponding sequence $\mathbf{v}_0, \ldots, \mathbf{v}_k$ where $\mathbf{v}_0 = \vec{1}$ and $\mathbf{v}_i$ is equal either to $\mathcal{V}_{[p_i]}(\mathbf{v}_{i-1})$ or to $\mathbf{v}_{i-1}$, depending on whether $([p_i], L_i)$ is demonic or angelic, respectively. Here, $\mathcal{V}$ is the function defined in Section 3.2 (observe that the component $[p]$ of $\mathcal{D}_{A_L}$ containing p can be seen as a strongly connected demonic VASS after deleting all transitions from/to the states outside $[p]$). The vector $\mathbf{v}_k$ describes the maximal asymptotic growth of the counters achievable by Demon when Angel uses the simple locking strategy associated to the path. Furthermore, the sequence $\mathbf{v}_0, \ldots, \mathbf{v}_k$ is computable in time polynomial in $|\mathcal{A}|$ and all finite components of $\mathbf{v}_k$ are bounded by $2^{d \cdot |Q|}$ because the total number of all demonic $([p_i], L_i)$ in the path is bounded by $|Q|$ (cf. Proposition 8).

The problem whether $\mathcal{C}[c_i] \in \mathcal{O}(n^k)$ can be decided by an *alternating* polynomial-time algorithm which selects an initial vertex of the form $([p], \emptyset)$ universally, and then constructs a maximal path in $\mathcal{G}_\mathcal{A}$ from $([p], \emptyset)$ where the successors of demonic/angelic vertices are chosen universally/existentially, respectively. After obtaining a maximal path $([p_1], L_1), \dots, ([p_k], L_k)$, the vector $\mathbf{v}_k$ is computed in polynomial time, and the algorithm answers yes/no depending on whether $\mathbf{v}_k(i) \leq k$ or not, respectively. The problem whether $\mathcal{C}[c_i] \in \Omega(n^k)$ is decided similarly, but here the initial vertex is chosen existentially, the successors of demonic/angelic vertices are chosen existentially/universally, and the algorithm answers yes/no depending on whether $\mathbf{v}_k(i) \geq k$ or not, respectively. This proves the **PSPACE** upper bounds of Theorem 10.

Observe that the crucial parameter influencing the computational hardness of the asymptotic analysis for VASS games is the number of maximal paths in $\mathcal{G}_\mathcal{A}$. If $|Q_A|$ and $Deg([p], L)$ are bounded by constants, then the above alternating polynomial time algorithms can be simulated by *deterministic* polynomial time algorithms. Thus, we obtain the following:

▶ **Theorem 14.** *Let Λ be a class of VASS games such that for every $\mathcal{A} \in \Lambda$ we have that $|Q_A|$ and $Deg([p], L)$, where $([p], L)$ is a leaf of $\mathcal{G}_\mathcal{A}$, are bounded by a fixed constant depending only on Λ. Then, the problems whether $\mathcal{L}_\mathcal{A} \in \mathcal{O}(n^k)$, $\mathcal{L}_\mathcal{A} \in \Omega(n^k)$, $\mathcal{L}_\mathcal{A} \in \Theta(n^k)$ for given $\mathcal{A} \in \Lambda$ and $k \in \mathbb{N}$, are solvable in polynomial time (where the k is written in binary). The same results hold also for $\mathcal{C}[c]$ (for a given counter c of $\mathcal{A}$).*

5 Conclusions, future work

We presented a precise complexity classification for the problems of polynomial asymptotic complexity of demonic VASS and VASS games. We also identified the structural parameters making these problems computationally hard, and we indicated that these parameters may actually stay reasonably small when dealing with VASS abstractions of computer programs. The actual applicability and scalability of the presented results to the problems of program analysis requires a more detailed study including experimental evaluation.

From a theoretical point of view, a natural question is whether the scope of effective asymptotic analysis can be extended from purely non-deterministic VASS to VASS with probabilistic transitions (i.e., VASS MDPs and VASS stochastic games). These problems are challenging and motivated by their applicability to probabilistic program analysis.

References

1 M. Ajdarów and A. Kučera. Deciding polynomial termination complexity for VASS programs. *arXiv*, 2102.06889 [cs.LO], 2021. arXiv:2102.06889.

2 E. Albert, P. Arenas, S. Genaim, M. Gómez-Zamalloa, G. Puebla, D. V. Ramírez-Deantes, G. Román-Díez, and D. Zanardini. Termination and cost analysis with COSTA and its user interfaces. *ENTCS*, 258(1):109–121, 2009. doi:10.1016/j.entcs.2009.12.008.

3 C. Alias, A. Darte, P. Feautrier, and L. Gonnord. Multi-dimensional rankings, program termination, and complexity bounds of flowchart programs. In *Proceedings of SAS 2010*, volume 6337 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2010. doi:10.1007/978-3-642-15769-1_8.

4 T. Brázdil, K. Chatterjee, A. Kučera, P. Novotný, and D. Velan. Deciding fast termination for probabilistic VASS with nondeterminism. In *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings*, volume 11781 of *Lecture Notes in Computer Science*, pages 462–478. Springer, 2019. doi:10.1007/978-3-030-31784-3_27.

5 T. Brázdil, K. Chatterjee, A. Kučera, P. Novotný, D. Velan, and F. Zuleger. Efficient algorithms for asymptotic bounds on termination time in VASS. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 185–194. ACM, 2018. doi:10.1145/3209108.3209191.

6 M. Broy and M. Wirsing. On the algebraic specification of nondeterministic programming languages. In *CAAP '81, Trees in Algebra and Programming, 6th Colloquium, Genoa, Italy, March 5-7, 1981, Proceedings*, volume 112 of *Lecture Notes in Computer Science*, pages 162–179. Springer, 1981. doi:10.1007/3-540-10828-9_61.

7 Q. Carbonneaux, J. Hoffmann, T. W. Reps, and Z. Shao. Automated resource analysis with coq proof objects. In *Proceedings of CAV 2017*, volume 10427 of *Lecture Notes in Computer Science*, pages 64–85. Springer, 2017. doi:10.1007/978-3-319-63390-9_4.

8 W. Czerwiński, S. Lasota, R. Lazić, J. Leroux, and F. Mazowiecki. The reachability problem for Petri nets is not elementary. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing, STOC 2019, Phoenix, AZ, USA, June 23-26, 2019*, pages 24–33. ACM, 2019. doi:10.1145/3313276.3316369.

9 A. F.-Montoya and R. Hähnle. Resource analysis of complex programs with cost equations. In *Proceedings of APLAS 2014*, volume 8858 of *Lecture Notes in Computer Science*, pages 275–295. Springer, 2014. doi:10.1007/978-3-319-12736-1_15.

10 J. Giesl, C. Aschermann, M. Brockschmidt, F. Emmes, F. Frohn, C. Fuhs, J. Hensel, C. Otto, M. Plücker, P. Schneider-Kamp, T. Ströder, S. Swiderski, and R. Thiemann. Analyzing program termination and complexity automatically with aprove. *J. Autom. Reasoning*, 58(1):3–31, 2017. doi:10.1007/s10817-016-9388-y.

11 S. Gulwani, K. K. Mehra, and T. M. Chilimbi. SPEED: precise and efficient static estimation of program computational complexity. In *Proceedings of POPL 2009*, pages 127–139. ACM, 2009. doi:10.1145/1480881.1480898.

12 J. Hoffmann, K. Aehlig, and M. Hofmann. Multivariate amortized resource analysis. *ACM Trans. Program. Lang. Syst.*, 34(3):14:1–14:62, 2012. doi:10.1145/2362389.2362393.

13 A. Kučera, J. Leroux, and D. Velan. Efficient analysis of VASS termination complexity. In *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 676–688, 2020. doi:10.1145/3373718.3394751.

14 J. Leroux. Polynomial vector addition systems with states. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, volume 107 of *LIPIcs*, pages 134:1–134:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.134.

15 R. Lipton. The reachability problem requires exponential space. Technical report 62, Yale, 1976.

16 E.W. Mayr and A.R. Meyer. The complexity of the finite containment problem for Petri nets. *J. ACM*, 28(3):561–576, 1981. doi:10.1145/322261.322271.

17 Ch. Papadimitriou. *Computational Complexity.* Addison, 1994.

18 M. Sinn, F. Zuleger, and H. Veith. A simple and scalable static analysis for bound analysis and amortized complexity analysis. In *Proccedings of CAV 2014*, volume 8559 of *Lecture Notes in Computer Science*, pages 745–761. Springer, 2014. doi:10.1007/978-3-319-08867-9_50.

19 M. Sinn, F. Zuleger, and H. Veith. Complexity and resource bound analysis of imperative programs using difference constraints. *J. Autom. Reasoning*, 59(1):3–45, 2017. doi:10.1007/s10817-016-9402-4.

20 F. Zuleger. The polynomial complexity of vector addition systems with states. In *Foundations of Software Science and Computation Structures - 23rd International Conference, FOSSACS 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12077 of *Lecture Notes in Computer Science*, pages 622–641. Springer, 2020. doi:10.1007/978-3-030-45231-5_32.

Bisimulation by Partitioning Is $\Omega((m+n)\log n)$

Jan Friso Groote
Eindhoven University of Technology, The Netherlands

Jan Martens
Eindhoven University of Technology, The Netherlands

Erik de Vink
Eindhoven University of Technology, The Netherlands

Abstract

An asymptotic lowerbound of $\Omega((m+n)\log n)$ is established for partition refinement algorithms that decide bisimilarity on labeled transition systems. The lowerbound is obtained by subsequently analysing two families of deterministic transition systems – one with a growing action set and another with a fixed action set.

For deterministic transition systems with a one-letter action set, bisimilarity can be decided with fundamentally different techniques than partition refinement. In particular, Paige, Tarjan, and Bonic give a linear algorithm for this specific situation. We show, exploiting the concept of an oracle, that the approach of Paige, Tarjan, and Bonic is not of help to develop a generic algorithm for deciding bisimilarity on labeled transition systems that is faster than the established lowerbound of $\Omega((m+n)\log n)$.

2012 ACM Subject Classification Theory of computation; Theory of computation → Interactive computation; Theory of computation → Design and analysis of algorithms

Keywords and phrases Bisimilarity, partition refinement, labeled transition system, lowerbound

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.31

Funding *Jan Martens*: AVVA project NWO 612.001.751/TOP1.17.002.

Acknowledgements We are grateful to the anonymous reviewers of CONCUR 2021 their thorough reading and constructive feedback.

1 Introduction

Strong bisimulation [16, 13] is the gold standard for equivalence on labeled transition systems (LTSs). Deciding bisimulation equivalence among the states of an LTS is a crucial step for tool-supported analysis and model checking of LTSs. The well-known and widely-used partition refinement algorithm of Paige and Tarjan [14] has a worst-case upperbound $O(m\log n)$ for establishing the bisimulation equivalence classes. Here, m is the number of transitions and n is the number of states in an LTS. The algorithm of Paige and Tarjan seeks to find, starting from an initial partition, via refinement steps, the coarsest stable partition, that in fact is built from the bisimulation equivalence classes that are looked for. The algorithm achieves the complexity of the logarithm of the number of states n by restricting the amount of work for refining blocks and moving states. Refining blocks is carried out by only investigating the smaller splitting blocks, using an intricate bookkeeping trick. Only the smaller parts of a block that are to be moved to a new block are split off, leaving the bulk of the original block at its place. These specific ideas go back to [8] and make the difference with the earlier $O(mn)$ algorithm of Kanellakis and Smolka [11].

The Paige-Tarjan algorithm, with its format of successive refinements of an initial partition till a fixpoint is reached, has been leading for variations and generalizations for deciding specific forms of (strong) bisimilarities, see e.g. [4, 6, 7, 18, 10]. We are interested in the

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 31; pp. 31:1–31:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

question whether the Paige-Tarjan algorithm is computationally optimal. A result is provided by Berkholz et al. in a paper [2] that studies stable colourings of (coloured) graphs. More specifically, they show that for an undirected graph with n nodes and m edges, canonical coarsest bi-stable colouring is in $\Omega((m+n)\log n)$. Translated to LTSs, the result of [2] builds on the assumption that LTSs are essentially non-deterministic, i.e., every state has multiple outgoing transitions for the same label. A first contribution of the present paper is a lowerbound of the class of partition refinement algorithms for deciding bisimilarity of deterministic LTSs. We define what a partition refinement algorithm is and articulate the complexity in terms of the number of states that are moved. Then, a particular family of (deterministic) LTSs, called bisplitters, is shown to require $n\log n$ work. This strengthens the result of [2], actually answering an open question in it.

We obtain our lowerbound results assuming that algorithms use partition refinement. However, one may wonder if a different approach than partition refinement can lead to a faster decision procedure for bisimulation. For the specific case of deterministic LTSs with a singleton action set and a state labelling, Robert Paige, Robert Tarjan and Robert Bonic propose a linear algorithm [15], which we will refer to as Roberts' algorithm. In [5] it is proven that partition refinement à la Hopcroft has a lowerbound of $\Omega(n\log n)$ in this case. Concretely, this means that Roberts' algorithm achieves the essentially better performance by using a completely different technique than partition refinement to determine the bisimulation equivalence classes.

Crucial for Roberts' algorithm is the ability to identify, in linear time, the bisimilarity classes of cycles. In this paper we show that if the alphabet consists of at least two actions a rapid decision on "cycles" as in [15] will not be of help to improve on the Paige-Tarjan algorithm for general LTSs. We argue that the specialty in the algorithm of [15], viz. to be able to quickly decide the bisimilarity of the states on cycles, can be captured by means of a stronger notion, namely an oracle, that provides the bisimulation classes of the states of a so-called "end structure", the counterpart in the multiple action setting of a cycle in the single action setting. The oracle can be consulted to refine the initial partition with respect to the bisimilarity on the end structures of the LTS for free. We show that for the class of partition refinement algorithms enhanced with such an oracle, thus encompassing the algorithm of [15], the $n\log n$ lowerbound persists for non-degenerate action sets.

The family of $n\log n$-hard LTSs we use to establish the lowerbound, involve an action set of $\log n$ actions. Building on the two results already mentioned, and exploiting ideas borrowed from [15] to extend the bisimulation classes for the states in the end structures, i.e. cycles, to the states of the complete LTS, we provide another family of (deterministic) LTSs that have only two actions. Led by these LTSs we argue that for the two-action case the complexity of deciding bisimulation is $\Omega((m+n)\log n)$, whether we use an oracle or not.

The document is structured as follows. In Section 2 we give the necessary preliminaries on the problem. A recap of the linear algorithm of [15] is provided in Section 3. Next, we introduce the family of deterministic LTSs $\mathcal{B}_k$ for which we show in Section 4 that deciding bisimilarity is $\Omega(n\log n)$ for the class of partition refinement algorithms and for which we establish in Section 5 an $\Omega(n\log n)$ lowerbound for the class of partition refinement algorithms enhanced with an oracle for end structures. In Section 6 we introduce the family of deterministic LTSs $\mathcal{C}_k$, each involving two actions only, to take the number of transitions m into account and establish an $\Omega((m+n)\log n)$ lowerbound for partition refinement with and without oracle for end structures. We wrap up with concluding remarks.

2 Preliminaries

Given a set of states S, a *partition* of S is a set of sets of states $\pi \subseteq 2^S$ such that for all $B, B' \in \pi$ it holds that $B \neq \emptyset$, $B \cap B' = \emptyset$, and $\bigcup_{B\in\pi} B = S$. The elements of a partition are refered to as blocks. A partition π of S induces an equivalence relation $=_\pi \subseteq S \times S$, where for two states $s, t \in S$, $s =_\pi t$ iff the states are in the same block, i.e. there is a block $B \in \pi$ such that $s, t \in B$. A partition π of S is a *refinement* of a partition π' of S iff for every block $B \in \pi$ there is a block $B' \in \pi'$ such that $B \subseteq B'$. It follows that each block of π' is the union of blocks of π. The refinement is *strict* if $\pi \neq \pi'$. The common refinement of two partitions π and π' is the partition with blocks $\{\, B \cap B' \mid B \in \pi,\ B' \in \pi' \,\}$. A sequence of partitions $\pi_0, \ldots, \pi_n$ is called a refinement sequence iff π_{i+1} is a refinement of π_i, for all $0 \leqslant i < n$.

▶ **Definition 1.** *A labeled transition system with initial partition (LTS) $L = (S, \mathcal{A}, \rightarrow, \pi_0)$ is given by a finite set of states S, a finite alphabet of actions $\mathcal{A}$, a transition relation $\rightarrow \subseteq S \times \mathcal{A} \times S$, and a partition π_0 of S. A labeled transition system with initial partition is called deterministic (dLTS) if the transition relation is a total function $S \times \mathcal{A} \rightarrow S$.*

Note that we omit an initial state, as it is not relevant in this article. Note also that in the presence of an initial partition, an LTS with one action label represents a Kripke structure. For a dLTS with a set of states S and the initial partition $\pi_0 = \{S\}$ we have that π_0 itself already represents bisimilarity, contrary to LTSs in general.

Given an LTS $L = (S, \mathcal{A}, \rightarrow, \pi_0)$, states $s, t \in S$, and an action $a \in \mathcal{A}$, we write $s \xrightarrow{a} t$ instead of $(s, a, t) \in \rightarrow$. For dLTSs we occasionally write $L(s, a)$ for t, i.e., t is the image of the pair (s, a) of the function $\rightarrow$. We say that s *reaches* t via a iff $s \xrightarrow{a} t$. A state s reaches a set $U \subseteq S$ iff there is a state in U that is reached by s. A set of states $V \subseteq S$ is called *stable* under a set of states $U \subseteq S$ iff for all actions a either all states in V reach U via a, or no state in V reaches U via a. A partition π is stable under a set of states U iff each block $B \in \pi$ is stable under U. A partition π is called stable iff it is stable under all its blocks.

Following [16, 13], for an LTS L a symmetric relation $R \subseteq S \times S$ is called a bisimulation relation iff for all $(s, t) \in R$ and $a \in \mathcal{A}$, we have that $s \xrightarrow{a} s'$ for some $s' \in S$ implies that $t \xrightarrow{a} t'$ for some $t' \in S$ such that $(s', t') \in R$. In the setting of the present paper, as we incorporate the initial partition in the definition of an LTS, bisimilarity is slightly non-standard. For a bisimulation relation R, we additionally require that it respects the initial partition π_0 of L, i.e. $(s, t) \in R$ implies $s =_{\pi_0} t$. Two states $s, t \in S$ are called (strongly) bisimilar for L iff a bisimulation relation R exists with $(s, t) \in R$, notation $s \leftrightarrows_L t$. Bisimilarity is an equivalence relation on the set of states of L. We write $[s]_L^{\leftrightarrows}$ for the bisimulation equivalence class of the state s in L.

Partition refinement algorithms for deciding bisimilarity on LTSs start with an initial partition π_0, which is then repeatedly refined until a stable partition is reached. This stable partition is then the coarsest stable partition of the LTS refining π_0 and coincides with bisimilarity [11, 14].

We define that an algorithm is a partition refinement algorithm if it constructs a valid sequence of partitions. Concretely this means that a state s in a block B is only assigned to another block if there is reason to do so, i.e. there is a splitter block B' in the current partition to which s has a transition and some other state in B does not, or the other way around. Thus, the block B is not stable under the block B'. Moreover, we insist that each subsequent partition reflects some progress, i.e., π_{i+1} is a strict refinement of π_i. This leads to the following notion of a valid refinement and a valid partition sequence.

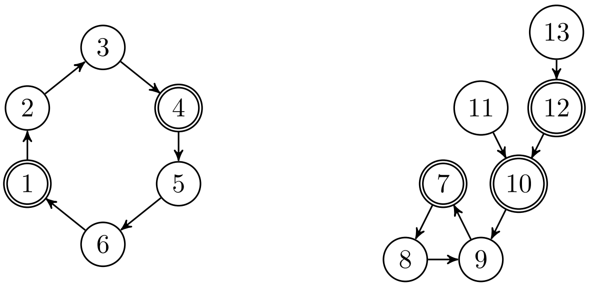


Figure 1 dLTS with one action label (not shown) and initial partition distinguishing states $1, 4, 7, 10, 12$ from states $2, 3, 5, 6, 8, 9, 11, 13$.

▶ **Definition 2.** *Let $L = (S, \mathcal{A}, \rightarrow, \pi_0)$ be an LTS, and π a partition of S. We call a refinement π' of π a* valid refinement *with respect to L, if the following criteria hold.*

(a) *π' is a strict refinement of π;*

(b) *if $s \neq_{\pi'} t$ for $s, t \in S$, then (i) $s \neq_\pi t$ or (ii) $s' \in S$ exists such that $s \xrightarrow{a} s'$ for some $a \in A$, and for all $t' \in S$ such that $t \xrightarrow{a} t'$ we have $s' \neq_\pi t'$, or the other way around with t replacing s.*

A sequence of partitions $\Pi = (\pi_0, \ldots, \pi_n)$ is called valid iff every successive partition π_i, for $0 < i \leqslant n$, is a valid refinement of π_{i-1}, and, moreover, the partition π_n is stable.

When a partition π is refined into a partition π', states that are in the same block but can reach different blocks can lead to a split of the block into smaller ones, each holding one of the states. This means that a block $B \in \pi$ is split into k blocks $B_1, \ldots, B_k \in \pi'$. The least amount of work is done for this operation, by creating new blocks for the least number of states possible. Thus, $B \in \pi$ is transformed into $B_1 \in \pi'$, say, the biggest block among $B_1, \ldots, B_k$. Therefore, the so-called refinement cost rc of the refinement π' of π is given by

$$rc(\pi, \pi') = \textstyle\sum_{B \in \pi} |B| - \max_{B' \in \pi' \colon B' \subseteq B} |B'| \,.$$

For a sequence of refinements $\Pi = (\pi_0, \ldots, \pi_n)$ we write $rc(\Pi)$ for $\sum_{i=1}^{n} rc(\pi_{i-1}, \pi_i)$. For an LTS L, we write $rc(L)$ for $\min\{\, rc(\Pi) \mid \Pi$ a valid refinement sequence for $L\,\}$. Note that this complexity measure is different from the one used in [2], which counts transitions. Our costs are always less or equal.

We characterise the states of LTSs by sequences of bits. The set of bits is denoted as $\mathbb{B} = \{\mathtt{0}, \mathtt{1}\}$. Bit sequences of length up to and including k are written as $\mathbb{B}^{\leqslant k}$. The inverse of a bit b is denoted by $\bar{b}$. Thus $\bar{\mathtt{0}} = \mathtt{1}$ and $\bar{\mathtt{1}} = \mathtt{0}$. For two bit sequences σ, σ', we write $\sigma \preccurlyeq \sigma'$ to indicate that σ is a prefix of σ' and write $\sigma \prec \sigma'$ if σ is a strict prefix of σ'. For a bit sequence $\sigma \in \mathbb{B}^k$, for any $i, j \leqslant k$ we write $\sigma[i]$ to indicate the bit at position i starting from position 1. We write $\sigma[i{:}j] = \sigma[i]\sigma[i{+}1] \cdots \sigma[j]$ to indicate the subword from position i to position j.

3 Roberts' algorithm

Most algorithms to determine bisimulation on an LTS use partition refinement. However, there is one notable exception. On the class of dLTSs with a singleton action alphabet, deciding the coarsest stable partition, i.e. bisimilarity, requires linear time only. This is due to the algorithm of Robert Paige, Robert Tarjan, and Robert Bonic [15], which we therefore aptly call Roberts' algorithm.

The algorithm exploits the structure of dLTSs with one label, of which examples are depicted in Figure 1 where the initial partition is indicated by single/double circles around the states. For each state in such a dLTS we can assign a unique cycle (also referred to as "end structure" in the sequel), and that state is either on this cycle or following outgoing edges lead to that (unique) cycle. Below, we roughly sketch how Roberts' algorithm works. See [15] for details.

1. Build lassos and mark states to identify the cycles of the dLTS.
2. Each state on a cycle encodes a sequence of states, viz. the states on the cycle in a specific order. The sequence of the blocks these states belong to forms a word (over the alphabet of the initial partition). Identify for each cycle the state with the lexicographic least such word. This can be done in linear time in the size of the cycle. If there are bisimilar states on the cycle, then the algorithm will identify them. States on different cycles can only be bisimilar if the number of non-bisimilar states on these cycles is the same. By comparing cycles with the same number of bisimulation equivalence classes, starting with the lexicographic least state, it is then determined linearly for all states on final cycles whether they are bisimilar.
3. By a backward calculation along the paths leading to the cycles the bisimilarity equivalence classes for the states not on cycles can then be determined in linear time as well.

A striking observation is that any partition refinement algorithm, using a valid sequence of partitions, requires a refinement cost of $\Omega(n \log n)$ to calculate which states are bisimilar for the dLTSs to which Roberts' algorithm applies. This follows from results in [3, 5] where it is shown that Hopcroft's algorithm [8] cannot have a better running time than $\Omega(n \log n)$. Below we come back to this observation, showing that the ideas in Roberts' algorithm cannot be exploited to come up with a linear algorithm for bisimulation if the LTS is either nondeterministic, or has more than one action label.

4 $\mathcal{B}_k$ is $\Omega(n \log n)$ for partition refinement

In this section we introduce a family of deterministic LTSs called bisplitters on which the cost of any partition refinement algorithm is $\Omega(n \log n)$ where n is the number of states. With some modification we obtain in Section 6 a family of LTSs that has the bound $\Omega((n+m) \log n)$ where m is the number of transitions.

▶ **Definition 3.** *For $k > 1$, the bisplitter $\mathcal{B}_k = (S, \mathcal{A}_k, \to, \pi_0)$ is defined as the dLTS that has the set $S = \{ \sigma \mid \sigma \in \mathbb{B}^k \}$ as its set of states, the set $\mathcal{A}_k = \{a_1, \ldots, a_{k-1}\}$ as its set of actions, the relation*

$$\{ \sigma \xrightarrow{a_i} \sigma \mid \sigma \in S,\ 1 \leqslant i < k \colon \sigma[i{+}1] = \mathtt{0} \} \cup \\ \{ \sigma \xrightarrow{a_i} \sigma[1{:}i{-}1]\,\overline{\sigma[i]}\,\mathtt{0}^{k-i} \mid \sigma \in S,\ 1 \leqslant i < k \colon \sigma[i{+}1] = \mathtt{1} \}$$

as its transition function, and the set $\pi_0 = \{ \{ \sigma \in S \mid \sigma[1] = \mathtt{0} \}, \{ \sigma \in S \mid \sigma[1] = \mathtt{1} \} \}$ as its initial partition.

Thus the bisplitter $\mathcal{B}_k$ has 2^k states, viz. the bitstrings of length k, and $\mathcal{B}_k$ has $k{-}1$ action labels. It has $(k{-}1)2^k$ transitions: (i) a self-loop for bitstring σ with label a_i if the $i{+}1$-th bit of σ equals 0; (ii) otherwise, i.e. when $i{+}1$-th bit of σ equals 1, the bitstring σ has for label a_i a transition to the bitstring that equals the first $i{-}1$ bits of σ, flips the i-th bit of σ, and has $k{-}i$-many 0's following. The initial partition π_0 distinguishes the bitstrings starting with 0 from those starting with 1. A drawing of bisplitter $\mathcal{B}_3$ is given in Figure 2. We see, e.g., for the bitstring $\sigma = \mathtt{101}$ an a_1-transition to itself, as $\sigma[2] = 0$, and an a_2-transition to 110, as $\sigma[3] = 1$.

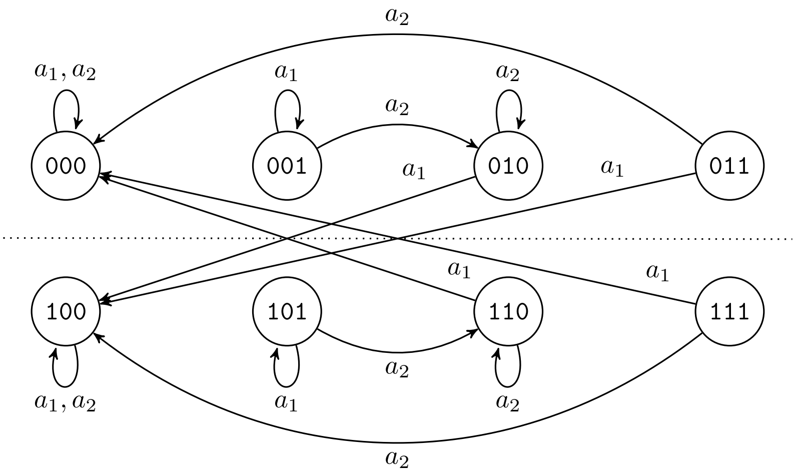


Figure 2 Bisplitter $\mathcal{B}_3$ with initial partition $\{\{000, 001, 010, 011\}, \{100, 101, 110, 111\}\}$.

▶ **Definition 4.** *For any string $\sigma \in \mathbb{B}^{\leqslant k}$, we define the prefix block B_σ of $\mathcal{B}_k$ to be the block $B_\sigma = \{\, \sigma' \in \mathbb{B}^k \mid \sigma \leqslant \sigma' \,\}$.*

The following lemma collects a number of results related to prefix blocks.

▶ **Lemma 5.** *Let $k \geqslant 2$ and let the dLTS $\mathcal{B}_k = (\mathbb{B}^k, \mathcal{A}_k, \rightarrow, \pi_0)$ be the k-th bisplitter. Let the sequence $\Pi = (\pi_0, \ldots, \pi_n)$ be a valid refinement sequence. Then it holds that*

(a) *Every partition π_i of Π contains prefix blocks only.*

(b) *If partition π_i of Π contains a prefix block B_σ with $|\sigma| < k$, then π_i is not stable.*

(c) *If B_σ is in π_i, for $0 \leqslant i < n$, then either $B_\sigma \in \pi_{i+1}$, or $B_{\sigma 1} \in \pi_{i+1}$ and $B_{\sigma 0} \in \pi_{i+1}$.*

Proof (a). Initially, for $\pi_0 = \{B_0, B_1\}$ both blocks are prefix blocks by definition. We prove, if partition π_i, for $0 \leqslant i < n$, has only prefix blocks then all blocks in π_{i+1} are prefix blocks as well.

Assume, to arrive at a contradiction, that there is a block $B \in \pi_{i+1}$ that is not a prefix block. Because π_{i+1} is a refinement of π_i, we have $B \subseteq B_\sigma$ for some prefix block $B_\sigma \in \pi_i$. This means that σ is a common prefix of all elements of B. We can choose θ such that $\sigma\theta$ is the longest common prefix of all elements of B. Since every singleton of $\mathbb{B}^k$ is a prefix block, B is not a singleton. This means that $|\sigma\theta| < k$, and that there are some elements σ_1 and σ_2 of B such that $\sigma\theta 0$ is a prefix of σ_1 and $\sigma\theta 1$ is a prefix of σ_2. Because B is not a prefix block, there must exist at least one $\tau \in \mathbb{B}^k$ with a prefix $\sigma\theta$ such that $\tau \notin B$. Obviously, we have either (i) $\sigma\theta 0$ is a prefix of τ, or (ii) $\sigma\theta 1$ is a prefix of τ. We will show that in both these cases τ in fact belongs to B in π_{i+1}, which is a contradiction. This also means that for any $B \in \pi_{i+1}$, where $B \subseteq B_\sigma$ for some prefix block $B_\sigma \in \pi_i$, we have that B is a prefix block for the prefix $\sigma\theta$ (i.e. $B = B_{\sigma\theta}$) where $\sigma\theta$ is the longest common prefix of all elements of B.

(i) Suppose $\sigma\theta 0$ is a prefix of τ. We will show that τ and σ_1 belong to the same block in π_{i+1} because for each a_j (where $1 \leqslant j < k$) the states σ_1' and τ', such that $\sigma_1 \xrightarrow{a_j} \sigma_1'$ and $\tau \xrightarrow{a_j} \tau'$, belong to the same block in π_i. There are three cases:

- $j < |\sigma\theta|$: Since $\sigma\theta$ is a prefix of both σ_1 and τ, we have $\sigma_1[j+1] = \tau[j+1]$.
  - If $\sigma_1[j+1] = \tau[j+1] = 0$, then $\sigma_1' = \sigma_1$ and $\tau' = \tau$. Obviously, both σ_1' and τ' belong to B_σ (since σ_1 and τ belong to B_σ).

  - If $\sigma_1[j+1] = \tau[j+1] = 1$, then both σ_1' and τ' are of the form $\rho[1 : j-1]\overline{\rho[j]}0^{k-j}$ where $\rho = \sigma\theta$, and we have $\sigma_1' = \tau'$, so they clearly belong to the same block of π_i.
- $j = |\sigma\theta|$: Since $\sigma_1[j+1] = \tau[j+1] = 0$, we have $\sigma_1' = \sigma_1$ and $\tau' = \tau$, and obviously both σ_1' and τ' belong to B_σ (since σ_1 and τ belong to B_σ).
- $j > |\sigma\theta|$: In fact, for arbitrary ρ, performing a_j with $j > |\sigma\theta|$ in $\sigma\theta\rho$ leads to $\sigma\theta\rho'$ (for both cases, where $(\sigma\theta\rho)[j+1]$ is 0 or 1). In particular this means that if $j > |\sigma\theta|$ and $\sigma_1 \xrightarrow{a_j} \sigma_1'$ and $\tau \xrightarrow{a_j} \tau'$, then $\sigma\theta$ is a prefix of both σ_1' and τ', and σ_1' and τ' belong to B_σ in π_i.

(ii) Now, suppose $\sigma\theta 1$ is a prefix of τ. We will show that τ and σ_2 belong to the same block in π_{i+1} because for each a_j (where $1 \leqslant j < k$) the states σ_2' and τ', such that $\sigma_2 \xrightarrow{a_j} \sigma_2'$ and $\tau \xrightarrow{a_j} \tau'$, belong to the same block in π_i. There are three cases:

- $j < |\sigma\theta|$: Similar as in (i).
- $j = |\sigma\theta|$: Since $\sigma_1[j+1] = \tau[j+1] = 1$, we have $\sigma_1' = \tau' = \rho[1 : j-1]\overline{\rho[j]}0^{k-1}$ where $\rho = \sigma\theta$, so clearly σ_1' and τ' are in a same block in π_i.
- $j > |\sigma\theta|$: Similar as in (i). ◀

Proof (b). Suppose $B_\sigma \in \pi_i$ and $|\sigma| = \ell < k$. Let $\theta \in \mathbb{B}^*$ be such that $\sigma_1 = \sigma 0\theta$ and $\sigma_2 = \sigma 1\theta$. Then we have $\sigma_1 \xrightarrow{a_\ell} \sigma_1 \in B_\sigma$ and $\sigma_2 \xrightarrow{a_\ell} \sigma[1{:}\ell-1]\overline{\sigma[\ell]}0^{k-\ell} \notin B_\sigma$. Thus B_σ isn't stable, and hence π_i isn't either. ◀

Proof (c). We show that for a prefix block $B_\sigma \in \pi_i$, a bit $b \in \mathbb{B}$ and all $\theta, \theta' \in \mathbb{B}^{k-(|\sigma|+1)}$ the states $\sigma_1 = \sigma b\theta$ and $\sigma_2 = \sigma b\theta'$ are not split by any action a_j, for $1 \leqslant j < k$, and thus are in the same block of π_{i+1}. Pick j, $1 \leqslant j < k$, and suppose $\sigma_1 \xrightarrow{a_j} \sigma_1'$, $\sigma_2 \xrightarrow{a_j} \sigma_2'$, i.e., $\sigma_1' = \mathcal{B}_k(\sigma_1, a_j)$ and $\sigma_2' = \mathcal{B}_k(\sigma_2, a_j)$. If $j \leqslant |\sigma|$ and $\sigma[j] = 0$ then $\sigma_1' = \sigma_1$ and $\sigma_2' = \sigma_2$ hence both $\sigma_1', \sigma_2' \in B_\sigma$ don't split for a_j. If $j \leqslant |\sigma|$ and $\sigma[j] = 1$ then $\sigma_1' = \sigma_2'$ and don't split for a_j either. If $j > |\sigma|$ then both $\sigma_1', \sigma_2' \in B_\sigma$ and don't split for a_j either. ◀

With the help of the above lemma, clarifying the form of the partitions in a valid refinement sequence for the bisplitter family, we are able to obtain a lowerbound for any partition refinement algorithm acting on it.

▶ **Theorem 6.** *For any $k > 1$, application of partition refinement to the bisplitter $\mathcal{B}_k$ has refinement costs $rc(\mathcal{B}_k) \in \Omega(n \log n)$ where $n = 2^k$ is the number of states of $\mathcal{B}_k$.*

Proof. Let $\Pi = \pi_0, \ldots, \pi_n$ be a valid refinement sequence for $\mathcal{B}_k$. By items a and b of Lemma 5, we have $\pi_n = \{\,\{s\} \mid s \in \mathbb{B}^k\,\}$ since π_n is stable. Item c of Lemma 5 implies that in every refinement step (π_i, π_{i+1}) a block is kept or it is refined in two blocks of equal size. The cost of refining the block B_σ, for $|\sigma| < k$, into $B_{\sigma 0}$ and $B_{\sigma 1}$ is the number of states in $B_{\sigma 0}$ or $B_{\sigma 1}$, which are the same and equal to $\frac{1}{2}2^{k-|\sigma|}$. Therefore, we have $rc(\mathcal{B}_k, \Pi) = \sum_{\ell=1}^{k-1} 2^\ell \frac{1}{2} 2^{k-\ell} = \sum_{\ell=1}^{k-1} \frac{1}{2} 2^k = (k-1)2^{k-1}$. If n is the number of states of $\mathcal{B}_k$, it holds that $n = 2^k$, thus $k - 1 = \log \frac{1}{2}n$. Hence, $rc(\mathcal{B}_k, \Pi) = \frac{1}{2}n \log \frac{1}{2}n$ which is in $\Omega(n \log n)$. ◀

5 $\mathcal{B}_k$ is $\Omega(n \log n)$ for partition refinement with an oracle

One may wonder whether the approach of calculating bisimulation equivalence classes will work on transition systems with non-degenerate action sets as well as the Roberts' algorithm guarantees a linear performance for the degenerate case. In order to capture the approach of [15], we augment the class of partition refinement algorithms with an oracle. At the start of the algorithm the oracle can be consulted to identify the bisimulation classes for designated states, viz. for those that are in an 'end structure', the counterpart of the cycles in [15]. This results in a refinement of the initial partition; partition refinement then starts from the updated partition.

Thus, we can ask the oracle to provide the bisimulation classes of all elements in an end structure of the LTS at hand. This yields a new partition, viz. the common refinement of the initial partition, on the one hand, and the partition induced by the bisimulation equivalence classes as given by the oracle and the complement of their union, on the other hand. The work that remains to be done is establishing the bisimulation equivalence classes, with respect to the initial partition, for the states not in any end structure. We will establish that a partition refinement algorithm strengthened with such an oracle will not improve upon partition refinement.

We first define the notion of an end structure of an LTS and the associated notion of an end structure partition.

▶ **Definition 7.** *Given an LTS $L = (S, \mathcal{A}, \to, \pi_0)$, a non-empty subset $S' \subseteq S$ is called an* end structure *of L, if S' is a minimimal set of states closed under all transitions. Moreover, $es(L) = \{\, S' \subseteq S \mid S'$ end structure of $L \,\}$ and $\pi_{es} = \{\, [s]^{\leftrightarrow}_{L} \mid s \in \bigcup es(L) \,\} \cup \{S \backslash \{\, [s]^{\leftrightarrow}_{L} \mid s \in \bigcup es(L) \,\}\} \setminus \{\emptyset\}$ is called the end structure partition of L.*

Like the cycles of [15], an LTS can have multiple end structures. The end structure partition π_{es} consists of the bisimilarity equivalence classes of L containing a state of an end structure, completed with a block of the remaining states that are not in an end structure, and not bisimilar to any state in an end structure (if not empty).

▶ **Lemma 8.** *Let $L = (S, \mathcal{A}, \to, \pi_0)$ be a dLTS.*
(a) *If $|\mathcal{A}| = 1$ then $es(L)$ consists of all cycles in L.*
(b) *Every $s \in S$ has a path to an end structure of L.*

Proof. (a) Since an end structure S' is closed under transitions, S' is a lasso. Because S' is minimal and non-empty, it follows that S' is a cycle.

(b) Let $U = \{\, t \in S \mid s \xrightarrow{w}{}^{*} t,\ w \in \mathcal{A}^* \,\}$ be the set of states reachable from state s. Then U is closed under all transitions. The minimal subset $U' \subseteq U$ which is still closed under all transitions is an end structure of L and reachable by s. ◀

Next we enhance the notion of a partition refinement algorithm. An oracle can be consulted for the states in the end structures. In this approach, the initial partition is replaced by a partition in which all bisimilarity equivalence classes of states in end structures are separated split off from the original blocks.

▶ **Definition 9.** *A partition refinement algorithm with end structure oracle yields for an LTS $L = (S, \mathcal{A}, \to, \pi_0)$ a valid refinement sequence $\Pi = (\pi'_0, \pi_1, \ldots, \pi_n)$ where π'_0 is the common refinement of the initial partition π_0 and the end structure partition π_{es} of L. The partition π'_0 is called the updated initial partition of L.*

As Roberts' algorithm shows, in the case of a singleton action set the availability of an end structure oracle yields the asymptotic performance of a linear algorithm. In the remainder of this section we confirm that in the case of more letters the end structure does not help either. The next lemma states that the amount of work required for the dLTS $\mathcal{B}_k$ by a partition refinement algorithm enhanced with an oracle dealing with end structures is at least the amount of work needed by a partition refinement algorithm without oracle for the dLTS $\mathcal{B}_{k-2}$.

▶ **Lemma 10.** *For the bisplitter $\mathcal{B}_k = (S, \mathcal{A}, \to, \pi_0)$, for some $k > 2$, let π_0' be the updated initial partition. Then every valid refinement sequence $\Pi = (\pi_0', \pi_2, \dots, \pi_n)$ for the updated bisplitter $\mathcal{B}_k' = (S, \mathcal{A}, \to, \pi_0')$ satisfies $rc(\Pi) \geqslant rc(\mathcal{B}_{k-2})$.*

Proof. Observe that there are only two end structures in $\mathcal{B}_k$, viz. the singletons of the two states with $\mathtt{0}^k$ and $\mathtt{1}\mathtt{0}^{k-1}$. Since all other states can reach $\mathtt{0}^k$ or $\mathtt{1}\mathtt{0}^{k-1}$, these states are not in an end structure: Choose $\sigma \in \mathbb{B}^k$, $\sigma \neq \mathtt{0}^k, \mathtt{1}\mathtt{0}^{k-1}$. Then σ is of the form $b\mathtt{0}^j\mathtt{1}\theta$ for some $b \in \mathbb{B}$, $j \geqslant 0$ and $\theta \in \mathbb{B}^*$. For $j = 0$ we have $\sigma \xrightarrow{a_1} \bar{b}\mathtt{0}^{k-1}$ which is either $\mathtt{0}^k$ or $\mathtt{1}\mathtt{0}^{k-1}$; for $j > 0$ we have $\sigma \xrightarrow{a_{j+1}} b\mathtt{0}^{j-1}\mathtt{1}\mathtt{0}^{k-(j+1)}$ while $b\mathtt{0}^{j-1}\mathtt{1}\mathtt{0}^{k-(j+1)}$ reaches $\mathtt{0}^k$ or $\mathtt{1}\mathtt{0}^{k-1}$ by induction.

By Lemma 5, every state $\sigma \in \mathbb{B}^k$ of $\mathcal{B}_k$ is in its own bisimulation equivalence class $\{\sigma\}$. It follows that the updated initial partition π_0' equals $\{\, \{\mathtt{0}^k\}, \{\mathtt{1}\mathtt{0}^{k-1}\}, B_0 \backslash \{\mathtt{0}^k\}, B_1 \backslash \{\mathtt{1}\mathtt{0}^{k-1}\} \,\}$. We claim that if a sequence $\Pi = (\pi_0', \pi_1, \dots, \pi_n)$ for $\mathcal{B}_k$ exists with costs $rc(\Pi) \leqslant rc(\mathcal{B}_{k-2})$, then also a valid refinement sequence Π' for $\mathcal{B}_{k-2}$ exists with costs smaller than $rc(\mathcal{B}_{k-2})$ which yields a contradiction, since, by definition, $rc(B_{k-2})$ are the minimum costs over all valid refinement sequence for $\mathcal{B}_{k-2}$.

So, assume $\Pi = (\pi_0', \pi_1, \dots \pi_n)$ is a valid refinement sequence for $\mathcal{B}_k$ and $rc(\Pi) \leqslant rc(\mathcal{B}_{k-2})$. We obtain a valid refinement sequence Π' for $\mathcal{B}_{k-2}$ in two steps. First, we use the projection function p from partitions on $\mathbb{B}^k$ to partitions on $\mathbb{B}_{k-2}$ that removes the prefix $\mathtt{11}$ from strings in a block (or ignores the block if such string is absent), i.e. $p(\pi) = \{\, \{\, \sigma \mid \mathtt{11}\sigma \in B \,\} \mid B \in \pi \,\} \backslash \{\emptyset\}$. In particular, $p(\pi_0) = \{\{\, \sigma \mid \sigma \in \mathbb{B}^{k-2} \,\}\}$. Second, the function P from refinement sequences of $\mathcal{B}_k$ to refinement sequences of $\mathcal{B}_{k-2}$, removes, in addition to application of p to each constituent partition, duplicate partitions from the sequence. Then $\Pi' = P(\Pi)$, say $\Pi' = (\varrho_0, \varrho_1, \dots, \varrho_\ell)$.

We have $\varrho_0 = p(\pi_0') = \{B_\varepsilon\}$. Next we observe that $\varrho_1 = \pi_0^{k-2} = \{B_0, B_1\}$ the initial partition of $\mathcal{B}_{k-2}$, containing the prefix blocks of $\mathtt{0}$ and $\mathtt{1}$: Take any two different states $b\theta, b\theta' \in \mathbb{B}^{k-2}$, for a bit $b \in \mathbb{B}$ and strings $\theta, \theta' \in \mathbb{B}^{k-3}$ that are not in the same block of ϱ_1. Let i, $0 \leqslant i < n$ be such that $p(\pi_i) = \varrho_0$ and $p(\pi_{i+1}) = \varrho_1$. Then $\mathtt{11}b\theta$ and $\mathtt{11}b\theta'$ have been separated when refining π_i into π_{i+1}. But no action a_j witnesses such a split: (i) $\mathcal{B}_k(\mathtt{11}b\theta, a_1) = \mathcal{B}_k(\mathtt{11}b\theta', a_1)$ as both equal $\mathtt{0}^k$; (ii) $\mathcal{B}_k(\mathtt{110}\theta, a_2) = \mathtt{110}\theta \in B_1 \backslash \{\mathtt{1}\mathtt{0}^{k-1}\}$ and $\mathcal{B}_k(\mathtt{110}\theta', a_2) = \mathtt{110}\theta' \in B_1 \backslash \{\mathtt{1}\mathtt{0}^{k-1}\}$; (iii) $\mathcal{B}_k(\mathtt{111}\theta, a_2) = \mathcal{B}_k(\mathtt{111}\theta', a_2)$, viz. are equal to 10^{k-1}; (iv) for $j > 2$ it holds that $\mathcal{B}_k(\mathtt{11}b\theta, a_j), \mathcal{B}_k(\mathtt{11}b\theta', a_j) \in B_1 \backslash \{\mathtt{1}\mathtt{0}^{k-1}\}$. Since $\varrho_1 \neq \varrho_0$, ϱ_1 has at least two blocks. Hence these must be B_0 and B_1.

Next we prove that every refinement of ϱ_i into ϱ_{i+1} of Π', for i, $1 \leqslant i < \ell$, is valid for $\mathcal{B}_{k-2}$. We first observe that, for all $\sigma, \sigma' \in \mathbb{B}^{k-2}$, $a_j \in \mathcal{A}$, it holds that $\mathcal{B}_{k-2}(\sigma, a_j) = \sigma'$ iff $\mathcal{B}_k(\mathtt{11}\sigma, a_{j+2}) = \mathtt{11}\sigma'$, a direct consequence of the definition of the transition functions of $\mathcal{B}_{k-2}$ and $\mathcal{B}_k$. From this we obtain

$$\sigma =_{\varrho_i} \sigma' \iff \mathtt{11}\sigma =_{\pi_h} \mathtt{11}\sigma' \tag{1}$$

provided $\varrho_i = p(\pi_h)$, for $0 \leqslant i \leqslant \ell$, via the definition of the projection function p. Now, consider subsequent partitions ϱ_i and ϱ_i in Π'. Let h, $0 \leqslant h < n$, be such that $\varrho_i = p(\pi_h)$ and $\varrho_{i+1} = p(\pi_{h+1})$. Clearly, ϱ_{i+1} is a refinement of ϱ_i; if for $B \in \pi_{h+1}$ we have $B = \bigcup_{\alpha \in I} B_\alpha$ with $B_\alpha \in \pi_h$ for $\alpha \in I$, then for $p[B] \in \varrho_{i+1}$ we have $p[B] = \bigcup_{\alpha \in I} p[B_\alpha]$ with $p[B_\alpha] \in \varrho_i$ for $\alpha \in I$. The validity of the refinement of ϱ_i into ϱ_{i+1} is justified by the validity of π_{h+1} into π_h. If $\sigma =_{\varrho_i} \sigma'$ and $\sigma \neq_{\varrho_{i+1}} \sigma'$ for $\sigma, \sigma' \in \mathbb{B}^{k-2}$, then $\sigma, \sigma' \in B_0$ or $\sigma, \sigma' \in B_1$ since ϱ_i is a refinement of ϱ_0. Moreover, $\mathtt{11}\sigma =_{\pi_h} \mathtt{11}\sigma'$ and $\mathtt{11}\sigma \neq_{\pi_{h+1}} \mathtt{11}\sigma'$ by (1). Hence, by validity, $\mathcal{B}_k(\mathtt{11}\sigma, a_j) \neq_{\pi_h} \mathcal{B}_k(\mathtt{11}\sigma', a_j)$. Clearly $j \neq 1$. Also, $j \neq 2$, since $\sigma[1] = \sigma'[1]$ we have $(\mathtt{11}\sigma)[3] = (\mathtt{11}\sigma')[3]$. Therefore, $\mathcal{B}_{k-2}(\sigma, a_{j-2}) \neq_{\pi_h} \mathcal{B}_{k-2}(\sigma', a_{j-2})$, showing the refinement of ϱ_i into ϱ_{i+1} to be valid.

Finally, since every block in π_n is a singleton, this is also the case for ϱ_ℓ. Thus, ϱ_ℓ is indeed the coarsest partition as required for Π' to be a valid refinement sequence for $\mathcal{B}_{k-2}$. Every refinement of ϱ_i into ϱ_{i+1} of Π' is projected from a refinement of some π_h into π_{h+1} of Π as argued above. Therefore, since $\varrho_i = p(\pi_i)$ and $\varrho_{i+1} = p(\pi_{i+1})$, we have $rc(\varrho_i, \varrho_{i+1}) \leqslant rc(\pi_h, \pi_{h+1})$, and hence $rc(\Pi) = \sum_{h=1}^{n} rc(\pi_{h-1}, \pi_h) \geqslant \sum_{i=}^{\ell} rc(\varrho_{i-1}, \varrho_i) = rc(\Pi') \geqslant rc(\mathcal{B}_{k-2})$, as was to be shown. ◀

Next we combine the above lemma with the lowerbound provided by Theorem 6 in order to prove the main result of this section.

▶ **Theorem 11.** *Any partition refinement algorithm with end structure oracle to decide bisimilarity for a dLTS is $\Omega(n \log n)$.*

Proof. Let $\mathcal{B}'_k$ be the updated bisplitter (with the initial partition π'_0 containing $\{0^k\}$, $B_0\backslash\{0^k\}$, $\{10^{k-1}\}$, and $B_1\backslash\{10^{k-1}\}$ as given by the oracle for end structures rather than the partition π_0 containing B_0 and B_1). By Lemma 10 we have, for $k > 2$, that $rc(\mathcal{B}'_k) \geqslant rc(\mathcal{B}_{k-2})$. By Theorem 6 we know that $rc(\mathcal{B}_{k-2}) \geqslant \frac{1}{2}n' \log \frac{1}{2}n'$ where $n' = 2^{k-2}$ is the number of states of $\mathcal{B}_{k-2}$. It holds that $n' = \frac{2^{k-2}}{2^k}n = \frac{1}{4}n$. So $rc(\mathcal{B}'_k) \geqslant \frac{1}{8}n \log \frac{1}{8}n$ from which we conclude that deciding bisimilarity for $\mathcal{B}_k$ with the help of an oracle for the end structures is $\Omega(n \log n)$. ◀

6 $\mathcal{C}_k$ is $\Omega((m+n)\log n)$ for partition refinement

We modify the bisplitter $\mathcal{B}_k$, that has an action alphabet of $k-1$ actions, to obtain a dLTS with two actions only. The resulting dLTS $\mathcal{C}_k$ has the action alphabet $\{a, b\}$, for each $k > 1$, and is referred to as the k-th *layered bisplitter*. We use $\mathcal{C}_k$ to obtain a $\Omega((n+m)\log n)$ lowerbound for deciding bisimilarity for LTSs with only two actions, where n is the number of states and m is de number of transitions.

To this end we adapt the construction of $\mathcal{B}_k$ at two places. Given an action alphabet $\mathcal{A}$ of $\mathcal{B}_k$ of $k-1$ actions, we introduce for each $\sigma \in \mathbb{B}^k$, a stake of 2^k states. Moreover, for each stake we add a tree gadget. These gadgets have height $\lceil \log(\frac{k-1}{2}) \rceil$ to accommodate $\lceil (k-1)/2 \rceil$ leaves.

▶ **Definition 12.** *Let $k > 1$, $\mathcal{B}_k$ be the k-th bisplitter, and $\mathbb{A} = \{a, b\}$. The dLTS $\mathcal{C}_k = (S_k^{\mathcal{C}}, \mathbb{A}, \rightarrow_{\mathcal{C}}, \pi_0^{\mathcal{C}})$, over the action set $\mathbb{A}$,*

(a) *has the set of states $S_k^{\mathcal{C}}$ defined as*

$$\begin{array}{rcl} S_k^{\mathcal{C}} & = & \{\, [\sigma, \ell] \in \mathbb{B}^k \times \mathbb{N} \mid 1 \leqslant \ell \leqslant 2^k \,\} \cup \\ & & \{\, \langle \sigma, w \rangle \in \mathbb{B}^k \times \mathbb{A}^* \mid 0 \leqslant |w| \leqslant \lceil \log(\frac{k-1}{2}) \rceil \,\}, \end{array}$$

(b) *has the transition relation $\rightarrow_{\mathcal{C}}$ given by*

$$\begin{array}{rcll} [\sigma, \ell] & \xrightarrow{\alpha}_{\mathcal{C}} & [\sigma, \ell+1] & \text{for } \sigma \in \mathbb{B}^k,\ 1 \leqslant \ell \leqslant 2^k,\ \alpha \in \mathbb{A} \\ [\sigma, 2^k] & \xrightarrow{\alpha}_{\mathcal{C}} & \langle \sigma, \varepsilon \rangle & \text{for } \sigma \in \mathbb{B}^k,\ \alpha \in \mathbb{A} \\ \langle \sigma, w \rangle & \xrightarrow{\alpha}_{\mathcal{C}} & \langle \sigma, w\alpha \rangle & \text{for } \sigma \in \mathbb{B}^k,\ |w| < \lceil \log(\frac{k-1}{2}) \rceil,\ \alpha \in \mathbb{A} \\ \langle \sigma, w \rangle & \xrightarrow{\alpha}_{\mathcal{C}} & [\sigma', 1] & \text{for } \sigma \in \mathbb{B}^k,\ |w| = \lceil \log(\frac{k-1}{2}) \rceil,\ \mathrm{bin}(w\alpha) = j,\ \mathcal{B}_k(\sigma, a_j) = \sigma', \end{array}$$

(c) *and has the initial partition $\pi_0^{\mathcal{C}}$ defined as*

$$\begin{array}{rcl} \pi_0^{\mathcal{C}} & = & \{\, \{\, [\sigma, \ell] \mid \sigma \in B_0 \,\}, \{\, [\sigma, \ell] \mid \sigma \in B_1 \,\} \bigm| 1 \leqslant \ell \leqslant 2^k \,\} \cup \\ & & \{\, \langle \sigma, w \rangle \in S_k^{\mathcal{C}} \mid \sigma \in \mathbb{B}^k,\ w \in \mathbb{A}^* \,\}. \end{array}$$

The auxilliary function bin : $\mathbb{A}^{\leqslant \lceil \log(k-1) \rceil} \rightarrow \mathbb{N}$, *used in item b is inductively defined by* $\mathrm{bin}(\varepsilon) = 0$, $\mathrm{bin}(wa) = \min\{2 * \mathrm{bin}(w), k-1\}$, *and* $\mathrm{bin}(wb) = \min\{2 * \mathrm{bin}(w)+1, k-1\}$.

We see that with each string $\sigma \in \mathbb{B}^k$ we associate in $\mathcal{C}_k$ as many as 2^k stake states $[\sigma, 1], \ldots, [\sigma, 2^k]$, one for each level ℓ, $1 \leqslant \ell \leqslant 2^k$. The stake states are traversed from the top $[\sigma, 1]$ to bottom $[\sigma, 2^k]$ on any string σ of length 2^k over $\mathbb{A}$. The tree gadget consists

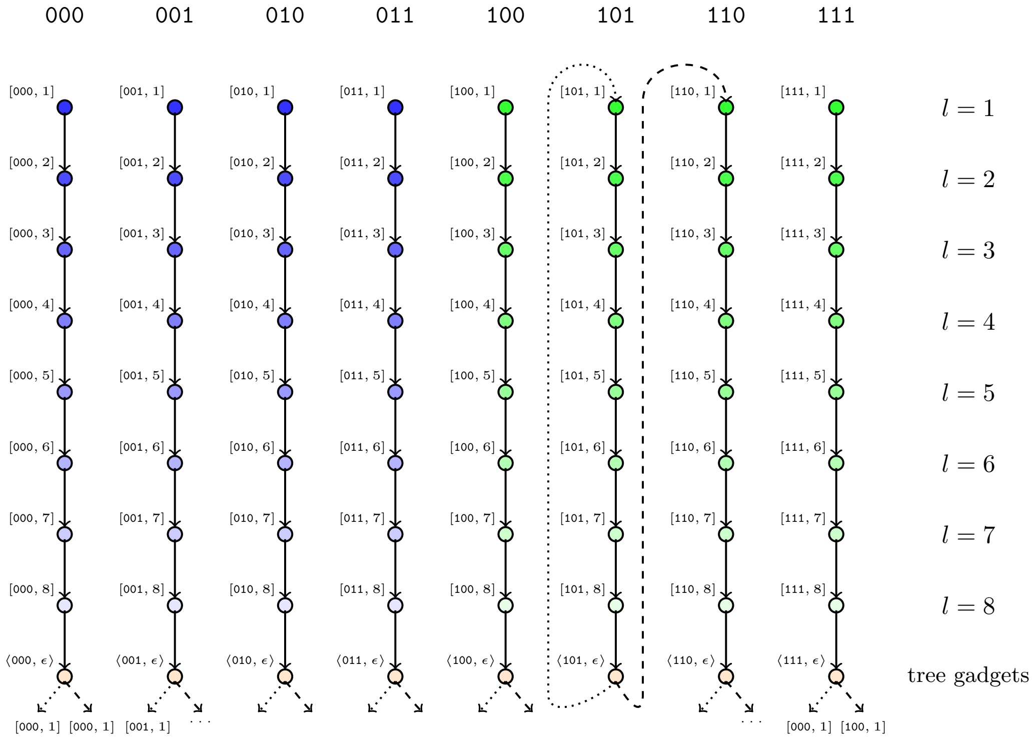


Figure 3 The partial layered bisplitter $\mathcal{C}_3$ with tree gadgets, the colors represent the initial partition.

of a complete binary tree of height $\lceil \log(\frac{k-1}{2}) \rceil$ that hence has at least $\lceil (k-1)/2 \rceil$ leaves. Traversal down the tree takes a left child on action a, a right child on action b. Together with the two actions of $\mathbb{A}$, $k-1$ source-label pairs can be encoded. To simulate a transition $\sigma \xrightarrow{a_j} \sigma'$ of $\mathcal{B}_k$ in $\mathcal{C}_k$ from a leaf of a tree gadget of σ to the top of the stake of σ', we need to be at a leaf $\langle \sigma, w \rangle$ of the tree gadget of σ such that the combined string $w\alpha$ for $\alpha \in \mathbb{A}$ is the binary encoding according of bin of the index j. An α-transition thus leads from the source $\langle \sigma, w \rangle$ to the target $[\sigma', 1]$ if $\sigma \xrightarrow{a_j} \sigma'$ in $\mathcal{B}_k$ and $w\alpha$ corresponds to j. The partition $\pi_0^{\mathcal{C}} = \{\, C_0^\ell, C_1^\ell \mid 1 \leqslant \ell \leqslant 2^k \,\} \cup \{C_\varepsilon\}$ distinguishes, for each level ℓ, the states at level ℓ of the stakes of strings starting with $\mathtt{0}$ in C_0^ℓ, the states of the stakes at level ℓ of strings starting with $\mathtt{1}$ in C_1^ℓ, and the states of the tree gadgets collected in C_ε.

Figure 3 depicts the two-label layered 3-splitter $\mathcal{C}_3$. Because also $\mathcal{B}_k$ has an action set of size 2 the tree gadgets only consist of the root node of the form $\langle \sigma, \varepsilon \rangle$. In Figure 2 of $\mathcal{B}_3$ we see that $\mathtt{101} \xrightarrow{a_1} \mathtt{101}$ and $\mathtt{101} \xrightarrow{a_2} \mathtt{110}$. In Figure 3 we have transitions $\langle \mathtt{101}, \varepsilon \rangle \xrightarrow{a} [\mathtt{101}, 1]$ and $\langle \mathtt{101}, \varepsilon \rangle \xrightarrow{b} [\mathtt{110}, 1]$ (dotted and dashed, respectively). Coloring of nodes is used to represent the initial partition $\pi_3^{\mathcal{C}}$ that separates 8 times, once for each level ℓ, the four states of the stakes in C_0^ℓ on the left from the four stake states in C_1^ℓ on the right, and the 8 tree states in C_ε at the bottom of the picture.

The 6-th bisplitter $\mathcal{B}_6$ has five actions, a_1 to a_5. A tree gadget for the layered bisplitter $\mathcal{C}_6$ with corresponding outgoing transitions is drawn in Figure 4. The tree has height $\lceil \log((6-1)/2) \rceil = \lceil \log \frac{5}{2} \rceil = 2$, hence it has $2^2 = 4$ leaves. Since each leaf has two outgoing transitions, one labeled a and one labeled b, the two leftmost leaves $\langle \sigma, aa \rangle$ and $\langle \sigma, ab \rangle$ are used with the two labels a and b to simulate transitions for a_1 up to a_4, the two rightmost leaves $\langle \sigma, ba \rangle$ and $\langle \sigma, bb \rangle$ have together four transitions all simulating the a_5-transition of σ.

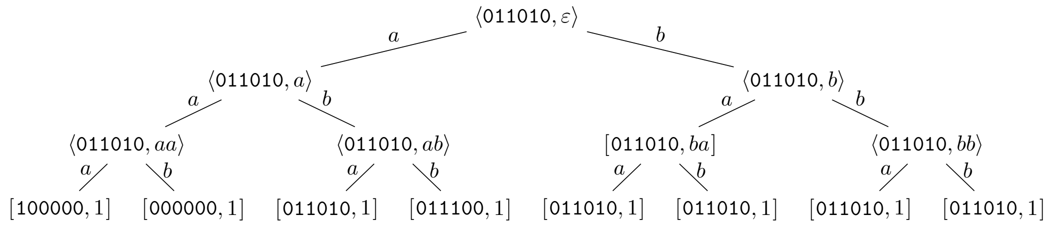


Figure 4 Example of the outgoing tree for $\mathcal{C}_6$ from the root $[\mathsf{011010}, \varepsilon] \in S_6^{\mathcal{C}}$.

The next lemma introduces three facts for the layered bisplitter $\mathcal{C}_k$ that we need in the sequel. The first states that if two states in different stakes, but at the same level, are separated during partition refinement, then all corresponding states at lower levels are separated as well. The second fact helps to transfer witnessing transitions in $\mathcal{B}_k$ to the setting of $\mathcal{C}_k$. A transition $\sigma \xrightarrow{a_j} \sigma'$ of $\mathcal{B}_k$ is reflected by a path from $[\sigma, 2^k]$ through the tree gadget of σ from root to leaf and then to the top state $[\sigma', 1]$ of the stake of σ'. The word $w\alpha$ encountered going down and out the tree gadget corresponds to the action a_j according to the bin-function. Lastly, it is shown that no two pairs of different states within the stakes are bisimilar.

▶ **Lemma 13.** *Let Π be a valid refinement sequence for $\mathcal{C}_k$ and π a partition in Π.*

(a) *If two states $[\sigma, \ell], [\sigma', \ell] \in S_k^{\mathcal{C}}$, for $1 \leqslant \ell \leqslant 2^k$, are in a different block of π, then all pairs $[\sigma, m], [\sigma', m] \in S$, for all levels m, $\ell \leqslant m \leqslant 2^k$, are in different blocks of π.*

(b) *If $[\sigma_1, 2^k]$ and $[\sigma_2, 2^k]$ are split in π, then exist $w \in \mathbb{A}^*$, $\alpha \in \mathbb{A}$, and $\sigma_1', \sigma_2' \in \mathbb{B}^k$ such that*

$$[\sigma_1, 2^k] \xrightarrow{w}{}^*_{\mathcal{C}} \langle \sigma_1, w \rangle \xrightarrow{\alpha}_{\mathcal{C}} [\sigma_1', 1] \quad \textit{and} \quad [\sigma_2, 2^k] \xrightarrow{w}{}^*_{\mathcal{C}} \langle \sigma_2, w \rangle \xrightarrow{\alpha}_{\mathcal{C}} [\sigma_2', 1]$$

with $[\sigma_1', 1]$ and $[\sigma_2', 1]$ in different blocks of π.

(c) *If π is the last refinement in Π, it contains the singletons of $[\sigma, \ell]$ for $\sigma \in \mathbb{B}^k$ and $1 \leqslant \ell \leqslant 2^k$.*

Proof. (a) For a proof by contradiction, suppose the partition π is the first partition of Π that falsifies the statement of the lemma. So $\pi \neq \pi_0^{\mathcal{C}}$, since for the initial refinement $\pi_0^{\mathcal{C}}$ the statement holds. Thus, π is a refinement of a partition π' in Π. So, there are two states $[\sigma, \ell], [\sigma', \ell] \in S_k^{\mathcal{C}}$ in different blocks of π while the states $[\sigma, \ell{+}1], [\sigma', \ell{+}1]$ are in the same block of π and hence of π'. Since $[\sigma, \ell]$ and $[\sigma', \ell]$ only have transitions to $[\sigma, \ell{+}1]$ and $[\sigma', \ell{+}1]$, respectively, that are in the same block π', the refinement wouldn't have been valid. We conclude that no falsifying partition π in Π exists and that the lemma holds.

(b) We first prove, by induction on $|w|$, that if $\langle \sigma_1, w \rangle$ and $\langle \sigma_2, w \rangle$ are split in π, then exist $w \in \mathbb{A}^*$ and $\alpha \in \mathbb{A}$ such that $\langle \sigma_1, w \rangle \xrightarrow{v}{}^* \langle \sigma_1, wv \rangle \xrightarrow{\alpha} [\sigma_1', 1]$ and $\langle \sigma_2, w \rangle \xrightarrow{v}{}^* \langle \sigma_2, wv \rangle \xrightarrow{\alpha} [\sigma_2', 1]$ with $[\sigma_1', 1]$ and $[\sigma_2', 1]$ in different blocks of π. If w has maximal length, $|w| = \lceil \log(\frac{k-1}{2}) \rceil$ this is clear. If $\langle \sigma_1, w \rangle$ and $\langle \sigma, w \rangle$ are split, for $|w| < \lceil \log(k{-}1) \rceil - 1$, then either a-transitions or b-transitions lead to split states. By the induction hypothesis, suitable paths exists from the targets of such transitions. Adding the respective transition proves the induction hypothesis. Since $[\sigma_1, 2^k]$ and $[\sigma_2, 2^k]$ can only reach $\langle \sigma_1, \varepsilon \rangle$ and $\langle \sigma_2, \varepsilon \rangle$ the statement follows.

(c) Choose ℓ, $1 \leqslant \ell \leqslant 2^k$ and define the relation $R \subseteq S_k^{\mathcal{B}} \times S_k^{\mathcal{B}}$ such that $(\sigma_1, \sigma_2) \in R$ iff the stake states $[\sigma_1, \ell], [\sigma_2, \ell] \in S_k^{\mathcal{C}}$ are bisimilar for $\mathcal{C}_k$. We verify that R is a bisimulation relation for $\mathcal{B}_k$. Note, that R respects $\pi_k^{\mathcal{B}}$, the initial partifion of $\mathcal{B}_k$. Now, suppose $(\sigma_1, \sigma_2) \in R$ and $\sigma_1 \xrightarrow{a_j} \sigma_1'$ for some $a_j \in \mathcal{A}_k$ and $\sigma_1' \in S_k^{\mathcal{B}}$. By construction of $\mathcal{C}_k$ we have $[\sigma_1, \ell] \xrightarrow{a^{2^k-\ell}}{}^* [\sigma_1, 2^k] \xrightarrow{a} \langle \sigma_1, \varepsilon \rangle \xrightarrow{w}{}^* \langle \sigma_1, w \rangle \xrightarrow{\alpha} [\sigma_1', 1] \xrightarrow{a^{\ell-1}}{}^* [\sigma_1', \ell]$ where $\mathrm{bin}(w\alpha) = j$.

Since $[\sigma_1, \ell]$ and $[\sigma_2, \ell]$ are bisimilar in $\mathcal{C}_k$, it follows that a corresponding path $[\sigma_2, \ell] \rightarrow^* [\sigma_2', \ell]$ exists in $\mathcal{C}_k$ with $[\sigma_1', \ell]$ and $[\sigma_2', \ell]$ bisimilar in $\mathcal{C}_k$. From this we derive that $\sigma_2 \xrightarrow{a_j} \sigma_2'$ in $\mathcal{B}_k$ and $(\sigma_1', \sigma_2') \in R$. Hence, R is a bisimulation relation for $\mathcal{B}_k$ indeed. Now, bisimilarity of $\mathcal{B}_k$ is discrete. Thus, if two stake states $[\sigma_1, \ell]$ and $[\sigma_2, \ell]$ are bisimilar for $\mathcal{C}_k$, then σ_1 and σ_2 are bisimilar for $\mathcal{B}_k$ thus $\sigma_1 = \sigma_2$, and therefore $[\sigma_1, \ell] = [\sigma_2, \ell]$. ◀

The next lemma states that all the splitting of states $[\sigma, \ell] \in S^{\mathcal{C}}$ at some level ℓ has refinement costs that are at least that of $\mathcal{B}_k$.

▶ **Lemma 14.** *It holds that $rc(\mathcal{C}_k) \geqslant 2^k rc(\mathcal{B}_k)$ for all $k > 1$.*

Proof. Let $\Pi = (\pi_0^{\mathcal{C}}, \pi_1, \ldots, \pi_n)$ be a valid refinement sequence for $\mathcal{C}_k$. We show that for each level ℓ, the sequence Π induces a valid refinement sequence Π^ℓ for $\mathcal{B}_k$.

The mapping p_ℓ assigns to a partition π of $\mathcal{C}_k$ a partition $p_\ell(\pi)$ by putting

$$p_\ell(\pi) = \{\, \{\, \sigma \in \mathbb{B}^k \mid [\sigma, \ell] \in B \,\} \mid B \in \pi \,\} \setminus \{\emptyset\}.$$

The sequence $\Pi^\ell = (\pi_0^\ell, \ldots, \pi_m^\ell)$ is obtained from the sequence $(p_\ell(\pi_0^{\mathcal{C}}), p_\ell(\pi_1), \ldots, p_\ell(\pi_n))$ by removing possible duplicates. We verify that Π^ℓ is a valid refinement sequence for $\mathcal{B}_k$.

First, we check that π_i^ℓ is a refinement of π_{i-1}^ℓ, for $1 \leqslant i \leqslant m$. Choose such an index i arbitrary. Let the index h with $1 \leqslant h \leqslant n$ by such that $p_\ell(\pi_{h-1}) = \pi_{i-1}^\ell$ and $p_\ell(\pi_i) = \pi_h^\ell$. For each block $B' \in \pi_i^\ell$ exists a block $B \in \pi_h$ such that $B' = p_\ell(B)$. Since π_h is a refinement of π_{h-1}, thus $B = \bigcup_r B_r$ for suitable $B_r \in \pi_h$. Note, $p_\ell(B_r) \in p_\ell(\pi_{h-1})$ for each index r. We have $B' = \bigcup_r p_\ell(B_r)$ with $p_\ell(B_r) \in \pi_{i-1}^\ell$, and π_i^ℓ is a refinement of π_{i-1}^ℓ.

Next, we verify that Π^ℓ is a valid refinement sequence for $\mathcal{B}_k$. Suppose the state $\sigma_1, \sigma \in S_k^{\mathcal{B}}$ are split for the refinement of π_{i-1}^ℓ into π_i^ℓ. Then the states $[\sigma_1, \ell], [\sigma_2, \ell] \in S_k^{\mathcal{C}}$ are split for the refinement of a partition π_{h-1} into the partition π_h for a some index h, $1 \leqslant h \leqslant n$. Then either (i) $\ell = 2^k$ and $[\sigma_1, \ell]$ and $[\sigma_2, \ell]$ have α-transitions to different blocks, for some $\alpha \in \mathbb{A}$, or (ii) $\ell < 2^k$ and $[\sigma_1, \ell{+}1]$ and $[\sigma_2, \ell{+}1]$ are in different blocks of π_{h-1}. In the case of (ii), it follows by Lemma 13 that also $[\sigma_1, 2^k]$ and $[\sigma_2, 2^k]$ are in different blocks of π_{h-1}. Thus, for the refinement of some π_{g-1} into π_g, $1 \leqslant g \leqslant h \leqslant n$, splitted the two states $[\sigma_1, 2^k]$ and $[\sigma_2, 2^k]$. By Lemma 13 exist $w \in \mathbb{A}^*$, $\alpha \in \mathbb{A}$, and $\sigma_1', \sigma_2' \in \mathbb{B}^k$ such that

$$[\sigma_1, 2^k] \xrightarrow{w}{}^*_{\mathcal{C}} \langle \sigma_1, w \rangle \xrightarrow{\alpha}_{\mathcal{C}} [\sigma_1', 1] \quad \text{and} \quad [\sigma_2, 2^k] \xrightarrow{w}{}^*_{\mathcal{C}} \langle \sigma_2, w \rangle \xrightarrow{\alpha}_{\mathcal{C}} [\sigma_2', 1]$$

with $[\sigma_1', 1]$ and $[\sigma_2', 1]$ in different blocks of π_{g-1}. Hence, σ_1' and σ_2' are in different blocks of π_{i-1}^ℓ while $\sigma_1 \xrightarrow{a_j}_{\mathcal{B}} \sigma_1'$ and $\sigma_2 \xrightarrow{a_j}_{\mathcal{B}} \sigma_2'$ for $j = \mathrm{bin}(w\alpha)$, which justifies splitting σ_1 and σ_2 for π_i^ℓ. We conclude that Π^ℓ is a valid refinement sequence for $\mathcal{B}_k$.

We have established that if Π is a valid refinement sequence for $\mathcal{C}_k$, then Π^ℓ is a valid refinement sequence for $\mathcal{B}_k$. The sequence Π^ℓ is obtained from Π by sifting out the blocks of Π's partitions and removing repeated partitions. Therefore it holds that $rc(\Pi) \geqslant rc(\Pi^\ell)$. Since the mapping p_ℓ and $p_{\ell'}$ include pairswise distinct sets of stake states for $\ell \neq \ell'$, $1 \leqslant \ell \leqslant 2^k$, it follows that $rc(\Pi) \geqslant \sum_{\ell=1}^{2^k} rc(\Pi^\ell) \geqslant 2^k rc(\mathcal{B}_k)$. Taking the minimum over all valid refinement sequences for $\mathcal{C}_k$ we conclude that $rc(\mathcal{C}_k) \geqslant 2^k rc(\mathcal{B}_k)$ as was to be shown. ◀

With the above technical lemma in place, we are able to strengthen the $\Omega(n \log n)$ lowerbound of Theorem 6 to account for the number of transitions. The improved lowerbound is $\Omega((m + n) \log n)$, where m is the number of transitions and n the number of states.

▶ **Theorem 15.** *Deciding bisimilarity for dLTSs with a partition refinement algorithm is $\Omega((m + n) \log n)$, where n is the number of states and m is the number of transitions of the dLTS.*

Proof. For the bisplitter $\mathcal{B}_k$, we know by Theorem 6 that $rc(\mathcal{B}_K) \geqslant 2^{k-1}(k-1)$. Thus, by Lemma 14, we obtain $rc(\mathcal{C}_K) \geqslant 2^{2k-1}(k-1)$. In the case of $\mathcal{C}_k$ we have $n = 2^k(2^k + 2^{\lceil \log(k-1) \rceil} - 1)$ and $m = 2n$. Hence $n + m \in \Theta(2^{2k-1})$ and $\log n \in \Theta(k-1)$, from which it follows that $rc(\mathcal{C}_k) \in \Omega((m+n)\log n)$. ◀

Underlying the proof of a lowerbound for deciding bisimilarity for the family of layered bisplitters $\mathcal{C}_k$ is the observation that each $\mathcal{C}_k$ can be seen as 2^k stacked instances of the ordinary bisplitters $\mathcal{B}_k$, augmented with tree gadgets to handle transitions properly. The other essential ingredient for the proof of Theorem 15 is the complexity of deciding bisimilarity with a partition refinement algorithm on the $\mathcal{B}_k$ family. The same reasoning applies when considering partition refinement algorithms with an oracle for end structures from Section 5. Also with an oracle the lowerbound of $\Omega((m+n)\log n)$ remains.

▶ **Theorem 16.** *Any partition refinement algorithm with an oracle for end structures that decides bisimilarity for dLTSs is $\Omega((m+n)\log n)$.*

Proof sketch. The proof is similar to that of Lemma 10 and Theorem 15. Consider, for some $k > 2$, the layered bisplitter $\mathcal{C}_k$ having initial partition π_0. The dLTS $\mathcal{C}_k$ has two end structures, viz. the set $S_0 \subset S_k^{\mathcal{C}}$ containing the states of the stake and accompanying tree gadget $S_0 = \{\, [0^k, \ell] \mid 1 \leqslant \ell \leqslant 2^k \,\} \cup \{\, \langle 0^k, w \rangle \mid w \in \mathbb{A}^*,\ |w| \leqslant \lceil \log(\frac{k-1}{2}) \rceil \}$ for 0^k and a similar $S_1 \subseteq S_k^{\mathcal{C}}$ for 10^{k-1}. The sets S_0 and S_1 are minimally closed under the transitions of $\mathcal{C}_k$. Other states, on the stake or tree gadget for a string σ, have a path to these sets inherited from a path from σ to 0^k or 10^k in $\mathcal{B}_k$. The bisimulation classes S_0' and S_1', say, with respect to $S_k^{\mathcal{C}}$ rather than π_0, consist of S_0 and S_1 themselves plus a part of the tree gadgets for transitions in $\mathcal{C}_k$ leading to S_0 and S_1, respectively.

The update of the initial partition π_0 with oracle information, which concerns, ignoring the tree gadgets, the common refinement of the layers on $\{\, [\sigma, \ell] \mid \sigma \in B_0 \,\}$ and $\{\, [\sigma, \ell] \mid \sigma \in B_1 \,\}$ on the one hand, and (a trivial refinement of) the bisimulation classes S_0' and S_1' on the other hand, is therefore equal to π on the stakes (and generally finer on the tree gadgets).

Then every valid refinement sequence $\Pi = (\pi_0', \pi_2, \ldots, \pi_n)$ for the updated dLTS $\mathcal{C}_k' = (S, \mathcal{A}, \to, \pi_0')$ satisfies $rc(\Pi) \geqslant rc(\mathcal{C}_{k-2})$. Following the lines of the proof of Lemma 10, we can show that a valid refinement sequence Π for $\mathcal{C}_k$ with updated initial partition π_0' induces a valid refinement sequence Π' for $\mathcal{C}_{k-2}$.

The number of states in $\mathcal{C}_{k-2}$ is $\Theta(n)$ with n the number of states of $\mathcal{C}_k$, and number of transitions in $\mathcal{C}_{k-2}$ is $\Theta(m)$ with m the number of transitions of $\mathcal{C}_k$. Therefore, $rc(\Pi) \geqslant rc(\Pi') \geqslant rc(\mathcal{C}_{k-2})$, from which we derive that any partition refinement algorithm with oracle for end structures involves $\Theta((m+n)\log n)$ times moving a state for $\mathcal{C}_k$ and that hence the algorithm is $\Omega((m+n)\log n)$. ◀

7 Conclusion

We have shown that, even when restricted to deterministic LTSs, it is not possible to construct an algorithm based on partition refinement that is more efficient than $\Omega((m+n)\log n)$. This strengthens the result of [2]. The bound obtained is preserved even when the algorithm is extended with an oracle that can determine for specific states whether they are bisimilar or not in constant time. The oracle proof technique enabled us to show that the algorithmic ideas underlying Roberts' algorithm [15] for the one-letter alphabet case cannot be used to come up with a fundamentally faster enhanced partition refinement algorithm for bisimulation. Of course, this is not addressing a generic lower bound to decide bisimilarity on LTSs, nor proving the conjecture that the Paige-Tarjan algorithm is optimal for deciding bisimilarity.

It is conceivable that a more efficient algorithm exists that is not based on partitioning for bisimulation. However, as it stands, no techniques are known to prove such a generic algorithmic lowerbound, and all that that do exist make assumptions on allowed operations, such as the well-known lowerbound on sorting. Further investigations to obtain a more general lowerbound may strenghten the oracle used even further, such that a wider range of algorithms is covered.

For the parallel setting, where deciding bisimilarity can be done faster indeed, a similar dichotomy between the case of a single letter alphabet and of a multiple letter alphabet occurs. For LTSs with multiple action labels a linear algorithm is proposed in [12], whereas for dLTSs with one action label it is possible to calculate bisimulation in logarithmic time, cf. [9]. The question is raised in [17], if a sub-linear parallel solution exists at all. Since this problem in a general setting is known to be P-complete as shown in [1], it is generally believed that no logarithmic algorithm is possible. It is worthwhile to transfer the results of this paper to a parallel setting, in order to better understand whether it is possible to design parallel partition based algorithms for bisimulation on LTSs that have a sub-linear complexity.

References

1 J. Balcázar, J. Gabarro, and M. Santha. Deciding bisimilarity is P-complete. *Formal Aspects of Computing*, 4(1):638–648, 1992.

2 C. Berkholz, P. Bonsma, and M. Grohe. Tight lower and upper bounds for the complexity of canonical colour refinement. *Theory of Computing Systems*, 60(4):581–614, 2017.

3 J. Berstel and O. Carton. On the complexity of Hopcroft's state minimization algorithm. In M. Domaratzki et al., editor, *Proc. CIAA 2004*, pages 35–44. LNCS 3317, 2004.

4 P. Buchholz. Exact performance equivalence: An equivalence relation for stochastic automata. *Theoretical Computer Science*, 215:263–287, 1999. doi:10.1016/S0304-3975(98)00169-8.

5 G. Castiglione, A. Restivo, and M. Sciortino. Hopcroft's algorithm and cyclic automata. In C. Martín-Vide et al., editor, *Proc. LATA 2008*, pages 172–183. LNCS 5196, 2008.

6 A. Dovier, C. Piazza, and A. Policriti. An efficient algorithm for computing bisimulation equivalence. *Theoretical Computer Sciemce*, 311:221–256, 2004. doi:10.1016/S0304-3975(03)00361-X.

7 J.F. Groote, H.J. Rivera Verduzco, and E.P. de Vink. An efficient algorithm to determine probabilistic bisimulation. *Algorithms*, 11(9):131,1–22, 2018.

8 J. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In Z. Kohavi and A. Paz, editors, *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.

9 J. Jájá and Kwan Woo Ryu. An efficient parallel algorithm for the single function coarsest partition problem. *Theoretical Computer Science*, 129(2):293–307, 1994.

10 D.N. Jansen, J.F. Groote, J.J.A. Keiren, and A. Wijs. An $O(m \log n)$ algorithm for branching bisimilarity on labelled transition systems. In A. Biere and D. Parker, editors, *Proc. TACAS*, pages 3–20. LNCS 12079, 2020. doi:10.1007/978-3-030-45237-7_1.

11 P.C. Kanellakis and S.A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Information and Computation*, 86(1):43–68, 1990. doi:10.1016/0890-5401(90)90025-D.

12 J. Martens, J.F. Groote, L. van de Haak, P. Hijma, and A. Wijs. A linear parallel algorithm to compute bisimulation and relational coarsest partitions. In preparation.

13 R. Milner. *A Calculus of Communicating Systems*. LNCS 92, 1980. doi:10.1007/3-540-10235-3.

14 R. Paige and R.E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

15 R. Paige, R.E. Tarjan, and R. Bonic. A linear time solution to the single function coarsest partition problem. *Theoretical Computer Science*, 40:67–84, 1985.

16 D.M.R. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Proc. 5th GI-Conference on Theoretical Computer Science*, pages 167–183. LNCS 104, 1981. doi:10.1007/BFb0017309.

17 S. Rajasekaran and I. Lee. Parallel algorithms for relational coarsest partition problems. *IEEE Transactions on Parallel and Distrubuted Systems*, 9(7):687–699, 1998.

18 T. Wißmann, U. Dorsch, S. Milius, and L. Schröder. Efficient and modular coalgebraic partition refinement. *Logical Methods Computer Science*, 16(1), 2020. doi:10.23638/LMCS-16(1:8)2020.

Explaining Behavioural Inequivalence Generically in Quasilinear Time

Thorsten Wißmann
Radboud University, Nijmegen, The Netherlands

Stefan Milius
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Lutz Schröder
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

Abstract

We provide a generic algorithm for constructing formulae that distinguish behaviourally inequivalent states in systems of various transition types such as nondeterministic, probabilistic or weighted; genericity over the transition type is achieved by working with coalgebras for a set functor in the paradigm of universal coalgebra. For every behavioural equivalence class in a given system, we construct a formula which holds precisely at the states in that class. The algorithm instantiates to deterministic finite automata, transition systems, labelled Markov chains, and systems of many other types. The ambient logic is a modal logic featuring modalities that are generically extracted from the functor; these modalities can be systematically translated into custom sets of modalities in a postprocessing step. The new algorithm builds on an existing coalgebraic partition refinement algorithm. It runs in time $\mathcal{O}((m+n)\log n)$ on systems with n states and m transitions, and the same asymptotic bound applies to the dag size of the formulae it constructs. This improves the bounds on run time and formula size compared to previous algorithms even for previously known specific instances, viz. transition systems and Markov chains; in particular, the best previous bound for transition systems was $\mathcal{O}(mn)$.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Program analysis

Keywords and phrases bisimulation, partition refinement, modal logic, distinguishing formulae, coalgebra

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.32

Related Version *Full Version*: `https://arxiv.org/abs/2105.00669`

Funding *Thorsten Wißmann*: Work forms part of the NWO TOP project 612.001.852 and the DFG-funded project COAX (MI 717/5-2).
Stefan Milius: Work forms part of the DFG-funded project CoMoC (MI 717/7-1).
Lutz Schröder: Work forms part of the DFG-funded project CoMoC (SCHR 1118/15-1).

1 Introduction

For finite transition systems, the Hennessy-Milner theorem guarantees that two states are bisimilar if and only if they satisfy the same modal formulae. This implies that whenever two states are not bisimilar, then one can find a modal formula that holds at one of the states but not at the other. Such a formula explains the difference of the two states' behaviour and is thus usually called a *distinguishing formula* [13]. For example, in the transition system in Figure 1, the formula $\Box\Diamond\top$ distinguishes the states x and y because x satisfies $\Box\Diamond\top$ whereas y does not. Given two states in a finite transition system with n states and m transitions, the algorithm by Cleaveland [13] computes a distinguishing formula in time $\mathcal{O}(mn)$. The algorithm builds on the Kanellakis-Smolka partition refinement algorithm [28, 29], which computes the bisimilarity relation on a transition system within the same time bound.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 32; pp. 32:1–32:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

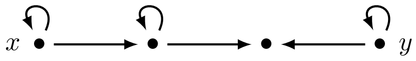


Figure 1 Example of a transition system.

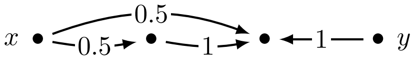


Figure 2 Example of a Markov chain.

Similar logical characterizations of bisimulation exist for other system types. For instance, Desharnais et al. [16, 17] characterize probabilistic bisimulation on (labelled) Markov chains, in the sense of Larsen and Skou [33] (for each label, every state has either no successors or a probability distribution on successors). In their logic, a formula $\Diamond_{\geq p}\phi$ holds at states that have a transition probability of at least p to states satisfying ϕ. For example, the state x in Figure 2 satisfies $\Diamond_{\geq 0.5}\Diamond_{\geq 1}\top$ but y does not. Desharnais et al. provide an algorithm that computes distinguishing formulae for labelled Markov chains in run time (roughly) $\mathcal{O}(n^4)$.

In the present work, we construct such counterexamples generically for a variety of system types. We achieve genericity over the system type by modelling state-based systems as coalgebras for a set functor in the framework of universal coalgebra [40]. Examples of coalgebras for a set functor include transition systems, deterministic automata, or weighted systems (e.g. Markov chains). Universal coalgebra provides a generic notion of behavioural equivalence that instantiates to standard notions for concrete system types, e.g. bisimilarity (transtion systems), language equivalence (deterministic automata), or probabilistic bisimilarity (Markov chains). Moreover, coalgebras come equipped with a generic notion of modal logic that is parametric in a choice of modalities whose semantics is constructed so as to guarantee invariance w.r.t. behavioural equivalence; under easily checked conditions, such a *coalgebraic modal logic* in fact characterizes behavioural equivalence in the same sense as Hennessy-Milner logic characterizes bisimilarity [39, 42]. Hence, as soon as suitable modal operators are found, coalgebraic modal formulae serve as distinguishing formulae.

In a nutshell, the contribution of the present paper is an algorithm that computes distinguishing formulae for behaviourally inequivalent states in *quasilinear time*, and in fact *certificates* that uniquely describe behavioural equivalence classes in a system, in coalgebraic generality. We build on an existing efficient coalgebraic partition refinement algorithm [46], thus achieving run time $\mathcal{O}(m \log n)$ on coalgebras with n states and m transitions (in a suitable encoding). The dag size of formulae is also $\mathcal{O}(m \log n)$ (for tree size, exponential lower bounds are known [22]); even for labelled transition systems, we thus improve the previous best bound $\mathcal{O}(mn)$ [13] for both run time and formula size. We systematically extract the requisite modalities from the functor at hand, requiring binary and nullary modalities in the general case, and then give a systematic method to translate these generic modal operators into more customary ones (such as the standard operators of Hennessy-Milner logic).

We subsequently identify a notion of *cancellative* functor that allows for additional optimization. E.g. functors modelling weighted systems are cancellative if and only if the weights come from a cancellative monoid, such as $(\mathbb{Z}, +)$, or $(\mathbb{R}, +)$ as used in probabilistic systems. For cancellative functors, much simpler distinguishing formulae can be constructed: the binary modalities can be replaced by unary ones, and only conjunction is needed in the propositional base. On labelled Markov chains, this complements the result that a logic with only conjunction and different unary modalities (mentioned above) suffices for the construction of distinguishing formulae (but not certificates) [17] (see also [19]).

Related Work. Cleaveland's algorithm [13] for labelled transition systems is is based on Kanellakis and Smolka's partition refinement algorithm [29]. The coalgebraic partition refinement algorithm we employ [46] is instead related to the more efficient Paige-Tarjan algorithm [36]. König et al. [32] extract formulae from winning strategies in a bisimulation game in coalgebraic generality; their algorithm runs in $\mathcal{O}(n^4)$ and does not support negative transition weights. Characteristic formulae for behavioural equivalence classes taken across *all* models require the use of fixpoint logics [21]. The mentioned algorithm by Desharnais et al. for distinguishing formulae on labelled Markov processes [17, Fig. 4] is based on Cleaveland's. No complexity analysis is made but the algorithm has four nested loops, so its run time is roughly $\mathcal{O}(n^4)$. Bernardo and Miculan [10] provide a similar algorithm for a logic with only disjunction. There are further generalizations along other axes, e.g. to behavioural preorders [12]. The TwoTowers tool set for the analysis of stochastic process algebras [8,9] computes distinguishing formulae for inequivalent processes, using variants of Cleaveland's algorithm. Some approaches construct alternative forms of certificates for inequivalence, such as Cranen et al.'s notion of evidence [14] or methods employed on business process models, based on model differences and event structures [5,6,18].

2 Preliminaries

We first recall some basic notation. We denote by $0 = \emptyset$, $1 = \{0\}$, $2 = \{0,1\}$ and $3 = \{0,1,2\}$ the sets representing the natural numbers 0, 1, 2 and 3. For every set X, there is a unique map $!\colon X \to 1$. We write Y^X for the set of functions $X \to Y$, so e.g. $X^2 \cong X \times X$. In particular, 2^X is the set of 2-valued *predicates* on X, which is in bijection with the *powerset* $\mathcal{P}X$ of X, i.e. the set of all subsets of X; in this bijection, a subset $A \in \mathcal{P}X$ corresponds to its *characteristic function* $\chi_A \in 2^X$, given by $\chi_A(x) = 1$ if $x \in A$, and $\chi(x) = 0$ otherwise. We generally indicate injective maps by $\rightarrowtail$. Given maps $f\colon Z \to X$, $g\colon Z \to Y$, we write $\langle f,g\rangle$ for the map $Z \to X \times Y$ given by $\langle f,g\rangle(z) = (f(z),g(z))$. We denote the disjoint union of sets X, Y by $X + Y$, with canonical inclusion maps $\mathsf{in}_1\colon X \rightarrowtail X + Y$ and $\mathsf{in}_2\colon Y \rightarrowtail X + Y$. More generally, we write $\coprod_{i\in I} X_i$ for the disjoint union of an I-indexed family of sets $(X_i)_{i\in I}$, and $\mathsf{in}_i\colon X_i \rightarrowtail \coprod_{i\in I} X_i$ for the i-th inclusion map. For a map $f\colon X \to Y$ (not necessarily surjective), we denote by $\ker(f) \subseteq X \times X$ the *kernel* of f, i.e. the equivalence relation

$$\ker(f) := \{(x,x') \in X \times X \mid f(x) = f(x')\}. \tag{1}$$

▶ **Notation 2.1** (Partitions)**.** Given an equivalence relation R on X, we write $[x]_R$ for the equivalence class $\{x' \in X \mid (x,x') \in R\}$ of $x \in X$. If R is the kernel of a map f, we simply write $[x]_f$ in lieu of $[x]_{\ker(f)}$. The intersection $R \cap S$ of equivalence relations is again an equivalence relation. The partition corresponding to R is denoted by $X/R = \{[x]_R \mid x \in X\}$. Note that $[-]_R\colon X \to X/R$ is a surjective map and that $R = \ker([-]_R)$.

A *signature* is a set Σ, whose elements are called *operation symbols*, equipped with a function $a\colon \Sigma \to \mathbb{N}$ assigning to each operation symbol its *arity*. We write $\sigma/n \in \Sigma$ for $\sigma \in \Sigma$ with $a(\sigma) = n$. We will apply the same terminology and notation to collections of modal operators.

2.1 Coalgebra

Universal coalgebra [40] provides a generic framework for the modelling and analysis of state-based systems. Its key abstraction is to parametrize notions and results over the transition type of systems, encapsulated as an endofunctor on a given base category. Instances cover, for example, deterministic automata, labelled (weighted) transition systems, and Markov chains.

▶ **Definition 2.2.** A *set functor* $F\colon \mathsf{Set} \to \mathsf{Set}$ assigns to every set X a set FX and to every map $f\colon X \to Y$ a map $Ff\colon FX \to FY$ such that identity maps and composition are preserved: $F\mathsf{id}_X = \mathsf{id}_{FX}$ and $F(g \cdot f) = Fg \cdot Ff$. An F*-coalgebra* is a pair (C,c) consisting of a set C (the *carrier*) and a map $c\colon C \to FC$ (the *structure*). When F is clear from the context, we simply speak of a *coalgebra*.

In a coalgebra $c\colon C \to FC$, we understand the carrier set C as consisting of *states*, and the structure c as assigning to each state $x \in C$ a structured collection of successor states, with the structure of collections determined by F. In this way, the notion of coalgebra subsumes numerous types of state-based systems, as illustrated next.

▶ **Example 2.3.**

1. The *powerset functor* $\mathcal{P}$ sends a set X to its powerset $\mathcal{P}X$ and a map $f\colon X \to Y$ to the map $\mathcal{P}f = f[-]\colon \mathcal{P}X \to \mathcal{P}Y$ taking direct images. A $\mathcal{P}$-coalgebra $c\colon C \to \mathcal{P}C$ is precisely a transition system: It assigns to every state $x \in C$ a set $c(x) \in \mathcal{P}C$ of *successor* states, inducing a transition relation $\to$ given by $x \to y$ iff $y \in c(x)$. Similarly, coalgebras for the finite powerset functor $\mathcal{P}_\mathsf{f}$ (with $\mathcal{P}_\mathsf{f}X$ being the set of finite subsets of X) are finitely branching transition systems.
2. Coalgebras for the functor $FX = 2 \times X^A$, where A is a fixed input alphabet, are deterministic automata (without an explicit initial state). Indeed, a coalgebra structure $c = \langle f, t\rangle\colon C \to 2 \times C^A$ consists of a finality predicate $f\colon C \to 2$ and a transition map $C \times A \to C$ in curried form $t\colon C \to C^A$.
3. Every signature Σ defines a *signature functor* that maps a set X to the set

   $$T_\Sigma X = \textstyle\coprod_{\sigma/n \in \Sigma} X^n,$$

   whose elements we may understand as flat Σ-terms $\sigma(x_1, \dots, x_n)$ with variables from X. The action of T_Σ on maps $f\colon X \to Y$ is then given by $(T_\Sigma f)(\sigma(x_1, \dots, x_n)) = \sigma(f(x_1), \dots, f(x_n))$. For simplicity, we write σ (instead of in_σ) for the coproduct injections, and Σ in lieu of T_Σ for the signature functor. States in Σ-coalgebras describe possibly infinite Σ-trees.
4. For a commutative monoid $(M, +, 0)$, the *monoid-valued functor* $M^{(-)}$ [25] is given by

   $$M^{(X)} := \{\mu\colon X \to M \mid \mu(x) = 0 \text{ for all but finitely many } x \in X\} \tag{2}$$

   on sets X; for a map $f\colon X \to Y$, the map $M^{(f)}\colon M^{(X)} \to M^{(Y)}$ is defined by

   $$(M^{(f)})(\mu)(y) = \textstyle\sum_{x \in X, f(x) = y} \mu(x).$$

   A coalgebra $c\colon C \to M^{(C)}$ is a finitely branching weighted transition system, where $c(x)(x') \in M$ is the transition weight from x to x'. For the Boolean monoid $\mathbb{B} = (2, \vee, 0)$, we recover $\mathcal{P}_\mathsf{f} = \mathbb{B}^{(-)}$. Coalgebras for $\mathbb{R}^{(-)}$, with $\mathbb{R}$ understood as the additive monoid of the reals, are $\mathbb{R}$-weighted transition systems. The functor

   $$\mathcal{D}X = \{\mu \in \mathbb{R}_{\geq 0}^{(X)} \mid \textstyle\sum_{x \in X} \mu(x) = 1\},$$

   which assigns to a set X the set of all finite probability distributions on X (represented as finitely supported probability mass functions), is a subfunctor of $\mathbb{R}^{(-)}$.
5. Functors can be composed; for instance, given a set A of labels, the composite of $\mathcal{P}$ and the functor $A \times (-)$ (whose action on sets maps a set X to the set $A \times X$) is the functor $FX = \mathcal{P}(A \times X)$, whose coalgebras are A-labelled transition systems. Coalgebras for $(\mathcal{D}(-) + 1)^A$ have been termed *probabilistic transition systems* [33] or *labelled Markov chains* [17], and coalgebras for $(\mathcal{D}((-) + 1) + 1)^A$ are *partial labelled Markov chains* [17]. Coalgebras for $SX = \mathcal{P}_\mathsf{f}(A \times \mathcal{D}X)$ are variously known as *simple Segala systems* or *Markov decision processes.*

We have a canonical notion of *behaviour* on F-coalgebras:

▶ **Definition 2.4.** An F-coalgebra *morphism* $h\colon (C,c) \to (D,d)$ is a map $h\colon C \to D$ such that $d \cdot h = Fh \cdot c$. States x, y in an F-coalgebra (C,c) are *behaviourally equivalent* ($x \sim y$) if there exists a coalgebra morphism h such that $h(x) = h(y)$.

$$\begin{tikzcd} C \arrow[r, "c"] \arrow[d, "h"'] & FC \arrow[d, "Fh"] \\ D \arrow[r, "d"] & FD \end{tikzcd}$$

Thus, we effectively define the behaviour of a state as those of its properties that are preserved by coalgebra morphisms. The notion of behavioural equivalence subsumes standard branching-time equivalences:

▶ **Example 2.5.**

1. For $F \in \{\mathcal{P}, \mathcal{P}_\mathsf{f}\}$, behavioural equivalence on F-coalgebras, i.e. on transition systems, is *bisimilarity* in the usual sense.
2. For deterministic automata as coalgebras for $FX = 2 \times X^A$, two states are behaviourally equivalent iff they accept the same formal language.
3. For a signature functor Σ, two states of a Σ-coalgebra are behaviourally equivalent iff they describe the same Σ-tree.
4. For labelled transition systems as coalgebras for $FX = \mathcal{P}(A \times X)$, coalgebraic behavioural equivalence precisely captures Milner's strong bisimilarity [1].
5. For weighted and probabilistic systems, coalgebraic behavioural equivalence instantiates to weighted and probabilistic bisimilarity, respectively [41, Cor. 4.7], [7, Thm. 4.2].

▶ **Remark 2.6.**

1. The notion of behavioural equivalence extends straightforwardly to states in different coalgebras, as one can canonically define the disjoint union of coalgebras.
2. We may assume without loss of generality that a set functor F preserves injective maps [43] (see also [2, 8.1.12–17]), that is, Ff is injective whenever f is.

2.2 Coalgebraic Logics

We briefly review basic concepts of coalgebraic modal logic [38,42]. Coalgebraic modal logics are parametric in a functor F determining the type of systems underlying the semantics, and additionally in a choice of modalities interpreted in terms of *predicate liftings*. For now, we use $F = \mathcal{P}$ as a basic example, deferring further examples to Section 5.

Syntax. The syntax of coalgebraic modal logic is parametrized over the choice of signature Λ of *modal operators* (with assigned arities). Then, *formulae* ϕ are generated by the grammar

$$\phi_1, \dots, \phi_n ::= \top \mid \neg\phi_1 \mid \phi_1 \wedge \phi_2 \mid \heartsuit(\phi_1, \dots, \phi_n) \qquad (\heartsuit/n \in \Lambda).$$

▶ **Example 2.7.** For $F = \mathcal{P}$, one often takes $\Lambda = \{\Diamond/1\}$; the induced syntax is that of (single-action) Hennessy-Milner logic. As usual, we write $\Box\phi :\equiv \neg\Diamond\neg\phi$.

Semantics. We interpret formulae as sets of states in F-coalgebras. This interpretation arises by assigning to each modal operator $\heartsuit/n \in \Lambda$ an n-ary *predicate lifting* $[\![\heartsuit]\!]$ [38,42], i.e. a family of maps $[\![\heartsuit]\!]_X\colon (2^X)^n \to 2^{FX}$, one for every set X, such that the *naturality* condition

$$Ff^{-1}\big[[\![\heartsuit]\!]_Y(P_1, \dots, P_n)\big] = [\![\heartsuit]\!]_X(f^{-1}[P_1], \dots, f^{-1}[P_n]) \tag{3}$$

for all $f\colon X \to Y$ and all $P_1, \dots, P_n \in 2^X$ (for categorically-minded readers, $[\![\heartsuit]\!]$ is a natural transformation $(2^{(-)})^n \to 2^{F^{\mathsf{op}}}$); the idea being to lift given predicates on states to predicates on structured collections of states. Given these data, the *extension* of a formula ϕ in an F-coalgebra (C,c) is a predicate $[\![\phi]\!]_{(C,c)}$, or just $[\![\phi]\!]$, on C, recursively defined by

$$[\![\top]\!]_{(C,c)} = C \qquad [\![\phi \wedge \psi]\!]_{(C,c)} = [\![\phi]\!]_{(C,c)} \cap [\![\psi]\!]_{(C,c)} \qquad [\![\neg\phi]\!]_{(C,c)} = C \setminus [\![\phi]\!]_{(C,c)}$$
$$[\![\heartsuit(\phi_1, \dots, \phi_n)]\!]_{(C,c)} = c^{-1}\big[[\![\heartsuit]\!]_C\big([\![\phi_1]\!]_{(C,c)}, \dots, [\![\phi_n]\!]_{(C,c)}\big)\big] \qquad (\heartsuit/n \in \Lambda)$$

(where we apply set operations to predicates with the evident meaning). We say that a state $x \in C$ *satisfies* ϕ if $[\![\phi]\!](x) = 1$. Notice how the clause for modalities says that x satisfies $\heartsuit(\phi_1, \dots, \phi_n)$ iff $c(x)$ satisfies the predicate obtained by lifting the predicates $[\![\phi_1]\!], \dots, [\![\phi_n]\!]$ on C to a predicate on FC according to $[\![\heartsuit]\!]$.

▶ **Example 2.8.** Over $F = \mathcal{P}$, we interpret $\Diamond$ by the predicate lifting

$$[\![\Diamond]\!]_X\colon 2^X \to 2^{\mathcal{P}X}, \quad P \mapsto \{K \subseteq X \mid \exists x \in K\colon x \in P\} = \{K \subseteq X \mid K \cap P \neq \emptyset\},$$

The arising notion of satisfaction over $\mathcal{P}$-coalgebras (C,c) is precisely the standard one: $x \in [\![\Diamond\phi]\!]_{(C,c)}$ iff $y \in [\![\phi]\!]_{(C,c)}$ for some transition $x \to y$.

The naturality condition (3) of predicate liftings guarantees invariance of the logic under coalgebra morphisms, and hence under behavioural equivalence:

▶ **Proposition 2.9** (Adequacy [38, 42]). *Behaviourally equivalent states satisfy the same formulae: $x \sim y$ implies that for all formulae ϕ, we have $x \in [\![\phi]\!]$ iff $y \in [\![\phi]\!]$.*

In our running example $F = \mathcal{P}$, this instantiates to the well-known fact that modal formulae are bisimulation-invariant, that is, bisimilar states in transition systems satisfy the same formulae of Hennessy-Milner logic.

3 Constructing Distinguishing Formulae

A proof method certifying behavioural equivalence of states x, y in a coalgebra is immediate by definition: One simply needs to exhibit a coalgebra morphism h such that $h(x) = h(y)$. In fact, for many system types, it suffices to relate x and y by a coalgebraic *bisimulation* in a suitable sense (e.g. [1, 24, 34, 40]), generalizing the Park-Milner bisimulation principle [35, 37]. It is less obvious how to certify behavioural *inequivalence* $x \not\sim y$, showing that such a morphism h does *not* exist. By Proposition 2.9, one option is to exhibit a (coalgebraic) modal formula ϕ that is satisfied by x but not by y. In the case of (image-finite) transition systems, such a formula is guaranteed to exist by the Hennessy-Milner theorem, which moreover is known to generalize to coalgebras [39, 42]. More generally, we consider separation of *sets* of states by formulae, following Cleaveland [13, Def. 2.4]:

▶ **Definition 3.1.** Let (C,c) be an F-coalgebra. A formula ϕ *distinguishes* a set $X \subseteq C$ from a set $Y \subseteq C$ if $X \subseteq [\![\phi]\!]$ and $Y \cap [\![\phi]\!] = \emptyset$. In case $X = \{x\}$ and $Y = \{y\}$, we just say that ϕ *distinguishes x from y*. We say that ϕ is a *certificate* of X if ϕ distinguishes X from $C \setminus X$, that is if $[\![\phi]\!] = X$.

Note that ϕ distinguishes X from Y iff $\neg\phi$ distinguishes Y from X. Certificates have also been referred to as *descriptions* [22]. If ϕ is a certificate of a behavioural equivalence class $[x]_\sim$, then by definition ϕ distinguishes x from y whenever $x \not\sim y$. To obtain distinguishing formulae for behaviourally inequivalent states in a coalgebra, it thus suffices to construct certificates for all behavioural equivalence classes, and our algorithm does just that. Of course, every certificate must be at least as large as a smallest distinguishing formula. However, already on transition systems, distinguishing formulae and certificates have the same asymptotic worst-case size (cf. Section 6).

A natural approach to computing certificates for behavioural equivalence classes is to extend algorithms that compute these equivalence classes. In particular, *partition refinement* algorithms compute a sequence $C/R_0, C/R_1, \dots$ of consecutively finer partitions (i.e. $R_{i+1} \subseteq R_i$) on the state space, where every *block* $B \in C/R_i$ is a union of behavioural equivalence classes, and the final partition is precisely $C/{\sim}$. Indeed, Cleaveland's algorithm for computing certificates on labelled transition systems [13] correspondingly extends Kanellakis and Smolka's partition refinement algorithm [28, 29], which runs in $\mathcal{O}(mn)$ on systems with $n = |C|$ states and m transitions. Our generic algorithm will be based on a more efficient partition refinement algorithm.

3.1 Paige-Tarjan with Certificates

Before we turn to constructing certificates in coalgebraic generality, we informally recall and extend the Paige-Tarjan algorithm [36], which computes the partition modulo bisimilarity of a given transition system with n states and m transitions in time $\mathcal{O}((m+n)\log n)$. We fix a given finite transition system, viewed as a $\mathcal{P}$-coalgebra $c\colon C \to \mathcal{P}C$.

The algorithm computes two sequences $(C/P_i)_{i\in\mathbb{N}}$ and $(C/Q_i)_{i\in\mathbb{N}}$ of partitions of C (with Q_i, P_i equivalence relations), where only the most recent partition is held in memory and i indexes the iterations of the main loop. Throughout the execution, C/P_i is finer than C/Q_i (that is, $P_i \subseteq Q_i$ for all i), and the algorithm terminates when $P_i = Q_i$. Intuitively, P_i is "one transition ahead" of Q_i: if Q_i distinguishes states x and y, then P_i is based on distinguishing transitions to x from transitions to y.

Initially, $C/Q_0 := \{C\}$ consists of only one block and C/P_0 of two blocks: the live states and the deadlocks (i.e. states with no outgoing transitions). If $P_i \subsetneqq Q_i$, then there is a block $B \in C/Q_i$ that is the union of at least two blocks in C/P_i. In such a situation, the algorithm chooses $S \subseteq B$ in C/P_i to have at most half the size of B and then splits the block B into S and $B \setminus S$ in the partition C/Q_i:

$$C/Q_{i+1} = (C/Q_i \setminus \{B\}) \;\cup\; \{S, B\setminus S\}.$$

This is correct because every state in S is already known to be behaviourally inequivalent to every state in $B \setminus S$. By the definition of bisimilarity, this implies that every block $T \in C/P_i$ with some transition to B may contain behaviourally inequivalent states as illustrated in Figure 3; that is, T may need to be split into smaller blocks, as follows:

(C1) states in T with successors in S but not in $B \setminus S$ (e.g. x_1 in Figure 3),
(C2) states in T with successors in S and $B \setminus S$ (e.g. x_2), and
(C3) states in T with successors $B \setminus S$ but not in S (e.g. x_3).

The partition C/P_{i+1} arises from C/P_i by splitting all such predecessor blocks T of B accordingly. If no such T is properly split, then $P_{i+1} = Q_{i+1}$, and the algorithm terminates. It is straightforward to construct certificates for the blocks arising during the execution:

- The certificate for the only block $C \in C/Q_0$ is $\top$, and the blocks for live states and deadlocks in C/P_0 have certificates $\Diamond\top$ and $\neg\Diamond\top$, respectively.
- In the refinement step, suppose that δ, β are certificates of $S \in C/P_i$ and $B \in C/Q_i$, respectively, where $S \subsetneqq B$. For every predecessor block T of B, the three blocks obtained by splitting T are distinguished (see Definition 3.1) as follows:

$$\text{(C1)}\quad \neg\Diamond(\beta\wedge\neg\delta), \qquad \text{(C2)}\quad \Diamond(\delta)\wedge\Diamond(\beta\wedge\neg\delta), \qquad \text{(C3)}\quad \neg\Diamond\delta. \tag{4}$$

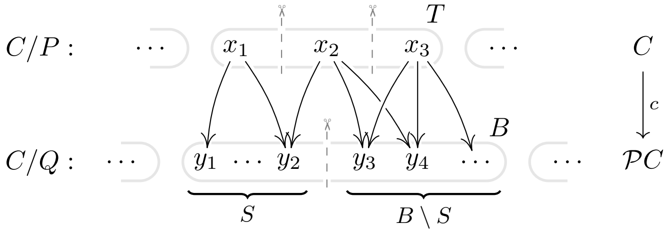


Figure 3 The refinement step as illustrated in [46, Figure 6].

Of course these formulae only distinguish the states in T from each other (e.g. there may be states in other blocks with transitions to both S and B). Hence, given a certificate ϕ of T, one obtains certificates of the three resulting blocks in C/P_{i+1} via conjunction: $\phi \wedge \neg\Diamond(\beta \wedge \neg\delta)$, etc.

Upon termination, every bisimilarity class $[x]_\sim$ in the transition system is annotated with a certificate. A key step in the generic development will be to come up with a coalgebraic generalization of the formulae for (C1)–(C3).

3.2 Generic Partition Refinement

The Paige-Tarjan algorithm has been adapted to other system types, e.g. weighted systems [44], and it has recently been generalized to coalgebras [20,46]. A crucial step in this generalization is to rephrase the case distinction (C1)–(C3) in terms of the functor $\mathcal{P}$: Given a predecessor block T in C/P_i for $S \subsetneqq B \in C/Q_i$, the three cases distinguish between the equivalence classes $[x]_{\mathcal{P}\chi_S^B \cdot c}$ for $x \in T$, where the map $\chi_S^B\colon C \to 3$ in the composite $\mathcal{P}\chi_S^B \cdot c\colon C \to \mathcal{P}3$ is defined by

$$\chi_S^B\colon C \to 3 \qquad \chi_S^B(x) = \begin{cases} 2 & \text{if } x \in S, \\ 1 & \text{if } x \in B \setminus S, \\ 0 & \text{if } x \in C \setminus B, \end{cases} \qquad \text{for sets } S \subseteq B \subseteq C. \tag{5}$$

Every case is a possible value of $t := \mathcal{P}\chi_S^B(c(x)) \in \mathcal{P}3$: (C1) $2 \in t \not\ni 1$, (C2) $2 \in t \ni 1$, and (C3) $2 \notin t \ni 1$. Since T is a predecessor block of B, the "fourth case" $2 \notin t \not\ni 1$ is not possible. There is a transition from x to some state outside of B iff $0 \in t$. However, because of the previous refinement steps performed by the algorithm, either all or no states states of T have an edge to $C \setminus B$ (a property called *stability* [36]), hence no distinction on $0 \in t$ is necessary.

It is now easy to generalize from transition systems to coalgebras by simply replacing the functor $\mathcal{P}$ with F in the refinement step. We recall the algorithm:

▶ **Algorithm 3.2** [46, Alg. 4.9, (5.1)]**.** Given a coalgebra $c\colon C \to FC$, put

$$C/Q_0 := \{C\} \qquad \text{and} \qquad P_0 := \ker(C \xrightarrow{c} FC \xrightarrow{F!} F1).$$

Starting at iteration $i = 0$, repeat the following while $P_i \neq Q_i$:

(A1) Pick $S \in C/P_i$ and $B \in C/Q_i$ such that $S \subsetneqq B$ and $2 \cdot |S| \leq |B|$

(A2) $C/Q_{i+1} := (C/Q_i \setminus \{B\}) \cup \{S, B \setminus S\}$

(A3) $P_{i+1} := P_i \cap \ker(\ C \xrightarrow{\ c\ } FC \xrightarrow{F\chi_S^B} F3\ )$

This algorithm formalizes the intuitive steps from Section 3.1. Again, two sequences of partitions P_1, Q_i are constructed, and $P_i = Q_i$ upon termination. Initially, Q_0 identifies all states and P_0 distinguishes states by only their output behaviour; e.g. for $F = \mathcal{P}$ and $x \in C$, the value $\mathcal{P}!(c(x)) \in \mathcal{P}1$ is $\emptyset$ if x is a deadlock, and $\{1\}$ if x is a live state, and for $FX = 2 \times X^A$, the value $F1(c(x)) \in F1 = 2 \times 1^A \cong 2$ indicates whether x is a final or non-final state.

In the main loop, blocks $S \in C/P_i$ and $B \in C/Q_i$ witnessing $P_i \subsetneqq Q_i$ are picked, and B is split into S and $B \setminus S$, like in the Paige-Tarjan algorithm. Note that step (A2) is equivalent to directly defining the equivalence relation Q_{i+1} as

$$Q_{i+1} := Q_i \cap \ker \chi_S^B.$$

A similar intersection of equivalence relations is performed in step (A3). The intersection splits every block $T \in C/P_i$ into smaller blocks such that $x, x' \in T$ end up in the same block iff $F\chi_S^B(c(x)) = F\chi_S^B(c(x'))$, i.e. T is replaced by $\{[x]_{F\chi_S^B(c(x))} \mid x \in T\}$. Again, this corresponds to the distinction of the three cases (C1)–(C3). For example, for $FX = 2 \times X^A$, there are $|F3| = 2 \cdot 3^{|A|}$ cases to be distinguished, and so every $T \in C/P_i$ is split into at most that many blocks.

The following property of F is needed for correctness [46, Ex. 5.11].

▶ **Definition 3.3** [46]. A functor F is *zippable* if map

$$\langle F(A+!), F(!+B)\rangle \colon\ F(A+B) \longrightarrow F(A+1) \times F(1+B)$$

is injective for all sets A, B.

Intuitively, $t \in F(A+B)$ is a term in variables from A and B. If F is zippable, then t is uniquely determined by the two elements in $F(A+1)$ and $F(1+B)$ obtained by identifying all B- and all A-variables with $0 \in 1$, respectively. E.g. $FX = X^2$ is zippable: $t = (\mathsf{in}_1(a), \mathsf{in}_2(b)) \in (A+B)^2$ is uniquely determined by $(\mathsf{in}_1(a), \mathsf{in}_2(0)) \in (A+1)^2$ and $(\mathsf{in}_1(0), \mathsf{in}_2(b)) \in (1+B)^2$, and similarly for the three other cases of t. In fact, all signature functors as well as $\mathcal{P}$ and all monoid-valued functors are zippable. Moreover, the class of zippable functors is closed under products, coproducts, and subfunctors but not under composition, e.g. $\mathcal{P}\mathcal{P}$ is not zippable [46].

▶ **Remark 3.4.** To apply the algorithm to coalgebras for composites FG of zippable functors, e.g. $\mathcal{P}(A \times (-))$, there is a reduction [46, Section 8] that embeds every FG-coalgebra into a coalgebra for the zippable functor $(F+G)(X) := FX + GX$. This reduction preserves and reflects behavioural equivalence, but introduces an intermediate state for every transition.

▶ **Theorem 3.5** [46, Thm 4.20, 5.20]. *On a finite coalgebra (C, c) for a zippable functor, Algorithm 3.2 terminates after $i \le |C|$ loop iterations, and the resulting partition identifies precisely the behaviourally equivalent states ($P_i = {\sim}$).*

3.3 Generic Modal Operators

The extended Paige-Tarjan algorithm (Section 3.1) constructs a distinguishing formula according to the three cases (C1)–(C3). In the coalgebraic Algorithm 3.2, these cases correspond to elements of $F3$, which determine in which block an element of a predecessor block T ends up. Indeed, the elements of $F3$ will also serve as generic modalities in characteristic formulae for blocks of states, essentially by the known equivalence between n-ary predicate liftings and (in this case, singleton) subsets of $F(2^n)$ [42] (termed *tests* by Klin [30]):

▶ **Definition 3.6.** The signature of $F3$-*modalities* for a functor F is

$$\Lambda = \{\ulcorner t \urcorner/2 \mid t \in F3\};$$

that is, we write $\ulcorner t \urcorner$ for the syntactic representation of a binary modality for every $t \in F3$. The interpretation of $\ulcorner t \urcorner$ for $F3$ is given by the predicate lifting

$$[\![\ulcorner t \urcorner]\!] \colon (2^X)^2 \to 2^{FX}, \qquad [\![\ulcorner t \urcorner]\!](S, B) = \{t' \in FX \mid F\chi^B_{S\cap B}(t') = t\}.$$

The intended use of $\ulcorner t \urcorner$ is as follows: Suppose a block B is split into subblocks $S \subseteq B$ and $B \setminus S$ with certificates δ and β, respectively: $[\![\delta]\!] = S$ and $[\![\beta]\!] = B$. As in Figure 3, we then split every predecessor block T of B into smaller parts, each of which is uniquely characterized by the formula $\ulcorner t \urcorner(\delta, \beta)$ for some $t \in F3$.

▶ **Example 3.7.** For $F = \mathcal{P}$, $\ulcorner\{0, 2\}\urcorner(\delta, \beta)$ is equivalent to $\overbrace{\Diamond\neg\beta}^{\text{“0”}} \wedge \neg\overbrace{\Diamond(\beta \wedge \neg\delta)}^{\text{“1”}} \wedge \overbrace{\Diamond(\delta \wedge \beta)}^{\text{“2”}}$.

▶ **Lemma 3.8.** *Given an F-coalgebra (C, c), $x \in C$, and formulae δ and β such that $[\![\delta]\!] \subseteq [\![\beta]\!] \subseteq C$, we have $x \in [\![\ulcorner t \urcorner(\delta, \beta)]\!]$ if and only if $F\chi^{[\![\beta]\!]}_{[\![\delta]\!]}(c(x)) = t$.*

In the initial partition C/P_0 on a transition system (C, c), we used the formulae $\Diamond\top$ and $\neg\Diamond\top$ to distinguish live states and deadlocks. In general, we can similarly describe the initial partition using modalities induced by elements of $F1$:

▶ **Notation 3.9.** Define the injective map $j_1 \colon 1 \rightarrowtail 3$ by $j_1(0) = 2$. Then the injection $Fj_1 \colon F1 \rightarrowtail F3$ provides a way to interpret elements $t \in F1$ as nullary modalities $\ulcorner t \urcorner$:

$$\ulcorner t \urcorner := \ulcorner Fj_1(t) \urcorner(\top, \top) \qquad \text{for } t \in F1.$$

(Alternatively, we could introduce $\ulcorner t \urcorner$ directly as a nullary modality.)

▶ **Lemma 3.10.** *For $x \in C$, $c \colon C \to FC$, and $t \in F1$, we have $x \in [\![\ulcorner t \urcorner]\!]$ if and only if $F!(c(x)) = t$.*

3.4 Algorithmic Construction of Certificates

The $F3$-modalities introduced above (Definition 3.6) induce an instance of coalgebraic modal logic (Section 2.2). We refer to coalgebraic modal formulae employing the $F3$-modalities as $F3$-*modal formulae*, and write $\mathcal{M}$ for the set of $F3$-modal formulae. As in the extended Paige-Tarjan algorithm (Section 3.1), we annotate every block arising during the execution of Algorithm 3.2 with a certificate in the shape of an $F3$-modal formula. Annotating blocks with formulae means that we construct maps

$$\beta_i \colon C/Q_i \to \mathcal{M} \qquad \text{and} \qquad \delta_i \colon C/P_i \to \mathcal{M} \qquad \text{for } i \in \mathbb{N}.$$

As in Algorithm 3.2, i indexes the loop iterations. For blocks B, S in the respective partition, $\beta_i(B)$, $\delta_i(S)$ denote corresponding certificates: we will have

$$\forall B \in X/Q_i \colon [\![\beta_i(B)]\!] = B \qquad \text{and} \qquad \forall S \in X/P_i \colon [\![\delta_i(S)]\!] = S. \tag{6}$$

We construct $\beta_i(B)$ and $\delta_i(S)$ iteratively, using certificates for the blocks $S \subsetneqq B$ at every iteration:

▶ **Algorithm 3.11.** We extend Algorithm 3.2 by the following. Initially, put

$$\beta_0(\{C\}) := \top \qquad \text{and} \qquad \delta_0([x]_{P_0}) := \ulcorner F!(c(x)) \urcorner \quad \text{for every } x \in C.$$

In the i-th iteration, extend steps (A2) and (A3) by the following assignments:

(A'2) $$\beta_{i+1}(D) \quad = \begin{cases} \delta_i(S) & \text{if } D = S \\ \beta_i(B) \wedge \neg\delta_i(S) & \text{if } D = B \setminus S \\ \beta_i(D) & \text{if } D \in C/Q_i \end{cases}$$

(A'3) $$\delta_{i+1}([x]_{P_{i+1}}) = \begin{cases} \delta_i([x]_{P_i}) & \text{if } [x]_{P_{i+1}} = [x]_{P_i} \\ \delta_i([x]_{P_i}) \wedge \ulcorner F\chi_S^B(c(x))\urcorner(\delta_i(S), \beta_i(B)) & \text{otherwise.} \end{cases}$$

Upon termination, return δ_i.

Like in Section 3.1, the only block of C/Q_0 has $\beta_0(\{C\}) = \top$ as a certificate. Since the partition C/P_0 distinguishes by the "output" (e.g. final vs. non-final states), the certificate of $[x]_{P_0}$ specifies what $F!(c(x)) \in F1$ is (Lemma 3.10).

In the i-th iteration of the main loop, we have certificates $\delta_i(S)$ and $\beta_i(B)$ for $S \subsetneq B$ in step (A1) satisfying (6) available from the previous iterations. In (A'2), the Boolean connectives describe how B is split into S and $B \setminus S$. In (A'3), new certificates are constructed for every predecessor block $T \in C/P_i$ that is refined. If T does not change, then neither does its certificate. Otherwise, the block $T = [x]_{P_i}$ is split into the blocks $[x]_{F\chi_S^B(c(x))}$ for $x \in T$ in step (A3), which is reflected by the $F3$ modality $\ulcorner F\chi_S^B(c(x))\urcorner$ as per Lemma 3.8.

▶ **Remark 3.12.** In step (A'2), $\beta_{i+1}(D)$ can be simplified to be no larger than $\delta_i(S)$. To see this, note that for $S \subseteq B \subseteq C$, $S \in X/P_i$, and $B \in X/Q_i$, every conjunct of $\beta_i(B)$ is also a conjunct of $\delta_i(S)$. In $\beta_i(B) \wedge \neg\delta_i(S)$, one can hence remove all conjuncts of $\beta_i(B)$ from $\delta_i(S)$, obtaining a formula δ', and then equivalently use $\beta_i(B) \wedge \neg\delta'$ in the definition of $\beta_{i+1}(D)$.

▶ **Theorem 3.13.** *For zippable F, Algorithm 3.11 is correct, i.e.* (6) *holds for all i. Thus, upon termination δ_i assigns certificates to each block of $C/{\sim} = C/P_i$.*

▶ **Corollary 3.14** (Hennessy-Milner)**.** *For zippable F, states x, y in a finite F-coalgebra are behaviourally equivalent iff they agree on all $F3$-modal formulae.*

▶ **Remark 3.15.** A smaller formula distinguishing a state x from a state y can be extracted from the certificates in time $\mathcal{O}(|C|)$. It is the leftmost conjunct that is different in the respective certificates of x and y. This is the subformula starting at the modal operator introduced in δ_i for the least i with $(x, y) \notin P_i$; hence, x satisfies $\ulcorner t\urcorner(\delta, \beta)$ but y satisfies $\ulcorner t'\urcorner(\delta, \beta)$ for some $t' \neq t$ in $F3$.

3.5 Complexity Analysis

The operations introduced by Algorithm 3.11 can be implemented with only constant run time overhead. To this end, one implements β and δ as arrays of formulae of length $|C|$ (note that at any point, there are at most $|C|$-many blocks). In the refinable-partition data structure [45], every block has an index (a natural number) and there is an array of length $|C|$ mapping every state $x \in C$ to the block it is contained in. Hence, for both partitions C/P and C/Q, one can look up a state's block and a block's certificate in constant time.

It is very likely that the certificates contain a particular subformula multiple times and that certificates of different blocks share common subformulae. For example, every certificate of a block refined in the i-th iteration using $S \subsetneq B$ contains the subformulas $\delta_i(S)$ and $\beta_i(B)$. Therefore, it is advantageous to represent all certificates constructed as one directed acyclic graph (dag) with nodes labelled by modal operators and conjunction and having precisely two outgoing edges. Moreover, edges have a binary flag indicating whether they represent negation $\neg$. Initially, there is only one node representing $\top$, and the operations of Algorithm 3.11 allocate new nodes and update the arrays for β and δ to point to the right nodes. For example, if the predecessor block $T \in C/P_i$ is refined in step (A'3), yielding a new block $[x]_{P_{i+1}}$, then a new node labelled $\wedge$ is allocated with edges to the nodes $\delta_i(T)$ and to another new node labelled $F\chi_S^B(c(x))$ with edges to the nodes $\delta_i(S)$ and $\delta_i(B)$.

For purposes of estimating the size of formulae generated by the algorithm, we use a notion of *transition* in coalgebras, inspired by the notion of canonical graph [26].

▶ **Definition 3.16.** *For states x, y in an F-coalgebra (C, c), we say that there is a* transition *$x \to y$ if $c(x) \in FC$ is not in the image $Fi[F(C \setminus \{y\})]$ $(\subseteq FC)$, where $i\colon C \setminus \{y\} \rightarrowtail C$ is the inclusion map.*

▶ **Theorem 3.17.** *For a coalgebra with n states and m transitions, the formula dag constructed by Algorithm 3.11 has size $\mathcal{O}(m \cdot \log n + n)$ and height at most $n + 1$.*

▶ **Theorem 3.18.** *Algorithm 3.11 adds only constant run time overhead, thus it has the same run time as Algorithm 3.2 (regardless of the optimization from Remark 3.12).*

For a tighter run time analysis of the underlying partition refinement algorithm, one additionally requires that F is equipped with a *refinement interface* [46, Def. 6.4], which is based on a given encoding of F-coalgebras in terms of *edges* between states (encodings serve only as data structures and have no direct semantic meaning, in particular do not entail a semantic reduction to relational structures). This notion of edge yields the same numbers (in $\mathcal{O}$-notation) as Definition 3.16 for all functors considered. All zippable functors we consider here have refinement interfaces [15, 46]. In presence of a refinement interface, step (A3) can be implemented efficiently, with resulting overall run time $\mathcal{O}((m + n) \cdot \log n \cdot p(c))$ where $n = |C|$, m is the number of edges in the encoding of the input coalgebra (C, c), and the *run-time factor* $p(c)$ is associated with the refinement interface. In most instances, e.g. for $\mathcal{P}$, $\mathbb{R}^{(-)}$, one has $p(c) = 1$; in particular, the generic algorithm recovers the run time of the Paige-Tarjan algorithm.

▶ **Remark 3.19.** The claimed run time relies on close attention to a number of implementation details. This includes use of an efficient data structure for the partition C/P_i [31, 45]; the other partition C/Q_i is only represented implicitly in terms of a queue of blocks $S \subsetneqq B$ witnessing $P_i \subsetneqq Q_i$, requiring additional care when splitting blocks in the queue [44, Fig. 3]. Moreover, grouping the elements of a block by $F3$ involves the consideration of a *possible majority candidate* [44].

▶ **Theorem 3.20.** *On a coalgebra with n states and m transitions for a zippable set functor with a refinement interface with factor $p(c)$, Algorithm 3.11 runs in time $\mathcal{O}((m+n) \cdot \log n \cdot p(c))$.*

4 Cancellative Functors

Our use of binary modalities relates to the fact that, as observed already by Paige and Tarjan, when splitting a block according to an existing partition of a block B into $S \subseteq B$ and $B \setminus S$, it is not in general sufficient to look only at the successors in S. However, this does suffice for some transition types; e.g. Hopcroft's algorithm for deterministic automata [27] and Valmari and Franceschinis' algorithm for weighted systems (e.g. Markov chains) [44] both split only with respect to S. In the following, we exhibit a criterion on the level of functors that captures that splitting w.r.t. only S is sufficient:

▶ **Definition 4.1.** A functor F is *cancellative* if the map

$$\langle F\chi_{\{1,2\}}, F\chi_{\{2\}} \rangle \colon F3 \to F2 \times F2$$

is injective.

To understand the role of the above map, recall the function $\chi_S^B \colon C \to 3$ from (5) and note that $\chi_{\{1,2\}} \cdot \chi_S^B = \chi_B$ and $\chi_{\{2\}} \cdot \chi_S^B = \chi_S$, so the composite $\langle F\chi_{\{1,2\}}, F\chi_{\{2\}}\rangle \cdot F\chi_S^B$ yields information about the accumulated transition weights into B and S but not about the one into $B \setminus S$; the injectivity condition means that for cancellative functors, this information suffices in the splitting step for $S \subseteq B \subseteq C$. The term *cancellative* stems from the respective property on monoids; recall that a monoid M is *cancellative* if $s + b_1 = s + b_2$ implies $b_1 = b_2$ for all $s, b_1, b_2 \in M$.

▶ **Proposition 4.2.** *The monoid-valued functor $M^{(-)}$ for a commutative monoid M is cancellative if and only if M is a cancellative monoid.*

Hence, $\mathbb{R}^{(-)}$ is cancellative, but $\mathcal{P}_\mathsf{f}$ is not. Moreover, all signature functors are cancellative:

▶ **Proposition 4.3.** *The class of cancellative functors contains the all constant functors as well as the identity functor, and it is closed under subfunctors, products, and coproducts.*

For example, $\mathcal{D}$ is cancellative, but $\mathcal{P}$ is not because of its subfunctor $\mathcal{P}_\mathsf{f}$.

▶ **Remark 4.4.** Cancellative functors are neither closed under quotients nor under composition. Zippability and cancellativity are independent properties. Zippability in conjunction with cancellativity implies m-zippability for all $m \in \mathbb{N}$, the m-ary variant [32] of zippability.

▶ **Theorem 4.5.** *If F is a cancellative functor, $\ulcorner F\chi_S^B(c(x))\urcorner(\delta_i(S), \beta_i(B))$ in Algorithm 3.11 can be replaced with $\ulcorner F\chi_S^C(c(x))\urcorner(\delta_i(S), \top)$. Then, the algorithm still correctly computes certificates in the given F-coalgebra (C, c).*

Note that in this optimized algorithm, the computation of β can be omitted because it is not used anymore. Hence, the resulting formulae only involve $\wedge$, $\top$, and modalities from the set $F3$ (with the second parameter fixed to $\top$). These modalities are equivalently unary modalities induced by elements of $F2$, which we term *$F2$-modalities*; hence, the corresponding Hennessy-Milner Theorem (Corollary 3.14) adapts to $F2$ for cancellative functors, as follows:

▶ **Corollary 4.6.** *For zippable and cancellative F, states in an F-coalgebra are behaviourally equivalent iff they agree on modal formulae built using $\top$, $\wedge$, and unary $F2$-modalities.*

5 Domain-Specific Certificates

On a given specific system type, one is typically interested in certificates and distinguishing formulae expressed via modalities whose use is established in the respective domain, e.g. $\Box$ and $\Diamond$ for transition systems. We next describe how the generic $F3$ modalities can be rewritten to domain-specific ones in a postprocessing step. The domain-specific modalities will not in general be equivalent to $F3$-modalities, but still yield certificates.

▶ **Definition 5.1.** The *Boolean closure* $\bar{\Lambda}$ of a modal signature Λ has as n-ary modalities propositional combinations of atoms of the form $\heartsuit(i_1, \dots, i_k)$, for $\heartsuit/k \in \Lambda$, where $i_1, \dots, i_k$ are propositional combinations of elements of $\{1, \dots, n\}$. Such a modality λ/n is interpreted by predicate liftings $[\![\lambda]\!]_X \colon (2^X)^n \to FX$ defined inductively in the obvious way.

For example, the boolean closure of $\Lambda = \{\Diamond/1\}$ contains the unary modality $\Box = \neg\Diamond\neg$.

▶ **Definition 5.2.** Given a modal signature Λ for a functor F, a *domain-specific interpretation* consists of functions $\tau \colon F1 \to \bar{\Lambda}$ and $\lambda \colon F3 \to \bar{\Lambda}$ assigning to each $o \in F1$ a nullary modality τ_o and to each $t \in F3$ a binary modality λ_t such that the predicate liftings $[\![\tau_o]\!]_X \in 2^{FX}$ and $[\![\lambda_t]\!]_X \colon (2^X)^2 \to 2^{FX}$ satisfy

$$[\![\tau_o]\!]_1 = \{o\} \quad (\text{in } 2^{F1}) \qquad \text{and} \qquad [t]_{F\chi_{\{1,2\}}} \cap [\![\lambda_t]\!]_3(\{2\}, \{1\}) = \{t\} \quad (\text{in } 2^{F3}).$$

(Recall that $\chi_{\{1,2\}}\colon 3 \to 2$ is the characteristic function of $\{1,2\} \subseteq 3$, and $[t]_{F\chi_{\{1,2\}}} \subseteq F3$ denotes the equivalence class of t w.r.t. $F\chi_{\{1,2\}}\colon F3 \to F2$.)

Thus, τ_o holds precisely at states with output behaviour $o \in F1$. Intuitively, $\lambda_t(\delta,\rho)$ describes the refinement step of a predecessor block T when splitting $B := [\![\delta]\!] \cup [\![\rho]\!]$ into $S := [\![\delta]\!]$ and $B \setminus S := [\![\rho]\!]$ (Figure 3), which translates into the arguments $\{2\}$ and $\{1\}$ of $[\![\lambda_t]\!]_3$. In the refinement step, we know from previous iterations that all elements have the same behaviour w.r.t. B. This is reflected in the intersection with $[t]_{F\chi_{\{1,2\}}}$. The axiom guarantees that λ_t characterizes $t \in F3$ uniquely, but only within the equivalence class representing a predecessor block. Thus, λ_t can be much smaller than equivalents of $\ulcorner t \urcorner$ (cf. Example 3.7):

▶ **Example 5.3.**

1. For $F = \mathcal{P}$, we have a domain-specific interpretation over the modal signature $\Lambda = \{\Diamond/1\}$. For $\emptyset, \{0\} \in \mathcal{P}1$, take $\tau_{\{0\}} = \Diamond\top$ and $\tau_\emptyset = \neg\Diamond\top$. For $t \in \mathcal{P}3$, we put

$$\begin{array}{llllll} \lambda_t(\delta,\rho) & = \neg\Diamond\rho & \text{if } 2 \in t \not\ni 1 & \lambda_t(\delta,\rho) & = \Diamond\delta \wedge \Diamond\rho & \text{if } 2 \in t \ni 1 \\ \lambda_t(\delta,\rho) & = \neg\Diamond\delta & \text{if } 2 \notin t \ni 1 & \lambda_t(\delta,\rho) & = \top & \text{if } 2 \notin t \not\ni 1. \end{array}$$

   The certificates obtained via this translation are precisely the ones generated in the example using the Paige-Tarjan algorithm, cf. (4), with ρ in lieu of $\beta \wedge \neg\delta$.

2. For a signature (functor) Σ, take $\Lambda = \{\sigma/0 \mid \sigma/n \in \Sigma\} \cup \{\langle{=}I\rangle/1 \mid I \in \mathcal{P}_{\mathsf{f}}(\mathbb{N})\}$. We interpret Λ by the predicate liftings

$$[\![\sigma]\!]_X = \{\sigma(x_1,\dots,x_n) \mid x_1,\dots,x_n \in X\} \subseteq \Sigma X,$$
$$[\![\langle{=}I\rangle]\!](S) = \{\sigma(x_1,\dots,x_n) \in \Sigma X \mid \forall i \in \mathbb{N}\colon i \in I \leftrightarrow (1 \le i \le n \ \wedge\ x_i \in S)\}.$$

   Intuitively, $\langle{=}I\rangle\,\phi$ states that the ith successor satisfies ϕ iff $i \in I$. We then have a domain-specific interpretation (τ,λ) given by $\tau_o := \sigma$ for $o = \sigma(0,\dots,0) \in \Sigma 1$ and $\lambda_t(\delta,\rho) := \langle{=}I\rangle\delta$ for $t = \sigma(x_1,\dots,x_n) \in \Sigma 3$ and $I = \{i \in \{1,\dots,n\} \mid x_i = 2\}$.

3. For a monoid-valued functor $M^{(-)}$, take $\Lambda = \{\langle{=}m\rangle/1 \mid m \in M\}$, interpreted by the predicate liftings $[\![\langle{=}m\rangle]\!]_X\colon 2^X \to 2^{M^{(X)}}$ given by

$$[\![\langle{=}m\rangle]\!]_X(S) = \{\mu \in M^{(X)} \mid m = \textstyle\sum_{x\in S}\mu(x)\}.$$

   A formula $\langle{=}m\rangle\,\delta$ thus states that the accumulated weight of the successors satisfying δ is exactly m. A domain-specific interpretation (τ,λ) is then given by $\tau_o = \langle{=}o(0)\rangle\,\top$ for $o \in M^{(1)}$ and $\lambda_t(\delta,\rho) = \langle{=}t(2)\rangle\,\delta \wedge \langle{=}t(1)\rangle\,\rho$ for $t \in M^{(3)}$. In case M is cancellative, we can also simply put $\lambda_t(\delta,\rho) = \langle{=}t(2)\rangle\,\delta$.

4. For labelled Markov chains, i.e. $FX = (\mathcal{D}X + 1)^A$, let $\Lambda = \{\langle a\rangle_p/1 \mid a \in A, p \in [0,1]\}$, where $\langle a\rangle_p\phi$ denotes that on input a, the next state will satisfy ϕ with probability at least p, as in cited work by Desharnais et al. [17]. This gives rise to the interpretation:

$$\tau_o = \bigwedge_{\substack{a\in A\\ o(a)\in\mathcal{D}1}} \langle a\rangle_1\top \wedge \bigwedge_{\substack{a\in A\\ o(a)\in 1}} \neg\langle a\rangle_1\top \qquad \lambda_t(\delta,\rho) = \bigwedge_{\substack{a\in A\\ t(a)\in\mathcal{D}3}} (\langle a\rangle_{t(a)(2)}\,\delta \wedge \langle a\rangle_{t(a)(1)}\,\rho)$$

Given a domain-specific interpretation (τ,λ) for a modal signature Λ for the functor F, we can postprocess certificates ϕ produced by Algorithm 3.11 by replacing the modalities $\ulcorner t \urcorner$ for $t \in F3$ according to the translation T recursively defined by the following clauses for modalities and by commutation with propositional operators:

$$T\big(\ulcorner t \urcorner(\top,\top)\big) = \tau_{F!(t)} \qquad T\big(\ulcorner t \urcorner(\delta,\beta)\big) = \lambda_t\big(T(\delta), T(\beta) \wedge \neg T(\delta)\big).$$

Note that one can replace $T(\beta) \wedge \neg T(\delta)$ with $T(\beta) \wedge \neg T(\delta')$ for the optimized δ' from Remark 3.12; the latter conjunction has essentially the same size as $T(\delta)$.

▶ **Proposition 5.4.** *For every certificate ϕ of a behavioural equivalence class of a given coalgebra produced by either Algorithm 3.11 or its optimization (Theorem 4.5), $T(\phi)$ is also a certificate for that class.*

Thus, the domain-specific modal signatures also inherit a Hennessy-Milner Theorem.

▶ **Example 5.5.** For labelled Markov chains ($FX = (\mathcal{D}X + 1)^A$) and the interpretation via the modalities $\langle a \rangle_p$ (Example 5.3.4), this yields certificates (thus in particular distinguishing formulae) in run time $\mathcal{O}(|A| \cdot m \cdot \log n)$, with the same bound on formula size. Desharnais et al. describe an algorithm [17, Fig. 4] that computes distinguishing formulae in the negation-free fragment of the same logic (they note also that this fragment does not suffice for certificates). They do not provide a run-time analysis, but the nested loop structure indicates that the asymptotic complexity is roughly $|A| \cdot n^4$.

6 Worst Case Tree Size of Certificates

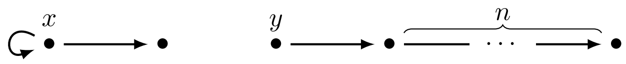
In the complexity analysis (Section 3.5), we have seen that certificates – and thus also distinguishing formulae – have dag size $\mathcal{O}(m \cdot \log n + n)$ on input coalgebras with n states and m transitions. However, when formulae are written in the usual linear way, multiple occurrences of the same subformula lead to an exponential blow up of the formula size in this sense, which for emphasis we refer to as the *tree size*.

Figueira and Gorín [22] show that exponential tree size is inevitable even for distinguishing formulae. The proof is based on winning strategies in bisimulation games, a technique that is also applied in other results on lower bounds on formula size [3, 4, 23].

▶ **Open Problem 6.1.** *Do states in $\mathbb{R}^{(-)}$-coalgebras generally have certificates of subexponential tree size in the number of states? If yes, can small certificates be computed efficiently?*

We note that for another cancellative functor, the answer is well-known: On deterministic automata, i.e. coalgebras for $FX = 2 \times X^A$, the standard minimization algorithm constructs distinguishing words of linear length.

▶ **Remark 6.2.** Cleaveland [13, p. 368] also mentions that minimal distinguishing formulae may be exponential in size, however for a slightly different notion of minimality: a formula ϕ distinguishing x from y is *minimal* if no ϕ obtained by replacing a non-trivial subformula of ϕ with the formula $\top$ distinguishes x from y. This is weaker than demanding that the formula size of ϕ is as small as possible. For example, in the transition system



for $n \in \mathbb{N}$,

the formula $\phi = \Diamond^{n+2}\top$ distinguishes x from y and is minimal in the above sense. However, x can in fact be distinguished from y in size $\mathcal{O}(1)$, by the formula $\Diamond\neg\Diamond\top$.

7 Conclusions and Further Work

We have presented a generic algorithm that computes distinguishing formulae for behaviourally inequivalent states in state-based systems of various types, cast as coalgebras for a functor capturing the system type. Our algorithm is based on coalgebraic partition refinement [46], and like that algorithm runs in time $\mathcal{O}((m+n) \cdot \log n \cdot p(c))$, with a functor-specific factor $p(c)$ that is 1 in many cases of interest. Independently of this factor, the distinguishing formulae constructed by the algorithm have dag size $\mathcal{O}(m \cdot \log n + n)$; they live in a dedicated instance of coalgebraic modal logic [39,42], with binary modalities extracted from the type functor in a systematic way. We have shown that for *cancellative* functors, the construction of formulae and, more importantly, the logic can be simplified, requiring only unary modalities and conjunction. We have also discussed how distinguishing formulae can be translated into a more familiar domain-specific syntax (e.g. Hennessy-Milner logic for transition systems).

There is an open source implementation of the underlying partition refinement algorithm [15], which may serve as a basis for a future implementation.

In partition refinement, blocks are successively refined in a top-down manner, and this is reflected by the use of conjunction in distinguishing formulae. Alternatively, bisimilarity may be computed bottom-up, as in a recent partition *aggregation* algorithm [11]. It is an interesting point for future investigation whether this algorithm can be extended to compute distinguishing formulae, which would likely be of a rather different shape than those computed via partition refinement.

References

1 Peter Aczel and Nax Mendler. A final coalgebra theorem. In *Proc. Category Theory and Computer Science (CTCS)*, volume 389 of *LNCS*, pages 357–365. Springer, 1989.

2 Jiří Adámek, Stefan Milius, and Lawrence S. Moss. Initial algebras, terminal coalgebras, and the theory of fixed points of functors. draft book, available online at `https://www8.cs.fau.de/ext/milius/publications/files/CoalgebraBook.pdf`, 2021.

3 Micah Adler and Neil Immerman. An *n!* lower bound on formula size. In *LICS 2001*, pages 197–206. IEEE Computer Society, 2001. `doi:10.1109/LICS.2001.932497`.

4 Micah Adler and Neil Immerman. An *n!* lower bound on formula size. *ACM Trans. Comput. Log.*, 4(3):296–314, 2003. `doi:10.1145/772062.772064`.

5 Abel Armas-Cervantes, Paolo Baldan, Marlon Dumas, and Luciano García-Bañuelos. Behavioral comparison of process models based on canonically reduced event structures. In *Business Process Management*, pages 267–282. Springer, 2014.

6 Abel Armas-Cervantes, Luciano García-Bañuelos, and Marlon Dumas. Event structures as a foundation for process model differencing, part 1: Acyclic processes. In *Web Services and Formal Methods*, pages 69–86. Springer, 2013.

7 Falk Bartels, Ana Sokolova, and Erik de Vink. A hierarchy of probabilistic system types. *Theoret. Comput. Sci.*, 327:3–22, 2004.

8 Marco Bernardo. TwoTowers 5.1 user manual, 2004.

9 Marco Bernardo, Rance Cleaveland, Steve Sims, and W. Stewart. TwoTowers: A tool integrating functional and performance analysis of concurrent systems. In *Formal Description Techniques and Protocol Specification, Testing and Verification, FORTE / PSTV 1998*, volume 135 of *IFIP Conference Proceedings*, pages 457–467. Kluwer, 1998.

10 Marco Bernardo and Marino Miculan. Constructive logical characterizations of bisimilarity for reactive probabilistic systems. *Theoretical Computer Science*, 764:80–99, 2019. Selected papers of ICTCS 2016.

11 Johanna Björklund and Loek Cleophas. Aggregation-based minimization of finite state automata. *Acta Informatica*, 2020.

12 Ufuk Celikkan and Rance Cleaveland. Generating diagnostic information for behavioral preorders. *Distributed Computing*, 9(2):61–75, 1995.

13 Rance Cleaveland. On automatically explaining bisimulation inequivalence. In *Computer-Aided Verification*, pages 364–372. Springer, 1991.

14 Sjoerd Cranen, Bas Luttik, and Tim A. C. Willemse. Evidence for Fixpoint Logic. In *24th EACSL Annual Conference on Computer Science Logic (CSL 2015)*, volume 41 of *LIPIcs*, pages 78–93. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015. `doi:10.4230/LIPIcs.CSL.2015.78`.

15 Hans-Peter Deifel, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Generic partition refinement and weighted tree automata. In *Formal Methods – The Next 30 Years, Proc. 3rd World Congress on Formal Methods (FM 2019)*, volume 11800 of *LNCS*, pages 280–297. Springer, October 2019.

16 J. Desharnais, A. Edalat, and P. Panangaden. A logical characterization of bisimulation for labeled markov processes. In *Proceedings. Thirteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.98CB36226)*, pages 478–487, 1998.

17 Josée Desharnais, Abbas Edalat, and Prakash Panangaden. Bisimulation for labelled markov processes. *Information and Computation*, 179(2):163–193, 2002.

18 Remco Dijkman. Diagnosing differences between business process models. In *Business Process Management*, pages 261–277, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

19 Ernst-Erich Doberkat. *Stochastic Coalgebraic Logic.* Springer, 2009. `doi:10.1007/978-3-642-02995-0`.

20 Ulrich Dorsch, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Efficient coalgebraic partition refinement. In *Proc. 28th International Conference on Concurrency Theory (CONCUR 2017)*, LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPIcs.CONCUR.2017.32`.

21 Ulrich Dorsch, Stefan Milius, Lutz Schröder, and Thorsten Wißmann. Predicate liftings and functor presentations in coalgebraic expression languages. In *Coalgebraic Methods in Computer Science, CMCS 2018*, volume 11202 of *LNCS*, pages 56–77. Springer, 2018. `doi:10.1007/978-3-030-00389-0_5`.

22 Santiago Figueira and Daniel Gorín. On the size of shortest modal descriptions. In *Advances in Modal Logic 8, papers from the eighth conference on "Advances in Modal Logic," held in Moscow, Russia, 24-27 August 2010*, pages 120–139. College Publications, 2010. URL: `http://www.aiml.net/volumes/volume8/Figueira-Gorin.pdf`.

23 Tim French, Wiebe van der Hoek, Petar Iliev, and Barteld Kooi. On the succinctness of some modal logics. *Artificial Intelligence*, 197:56–85, 2013.

24 Daniel Gorín and Lutz Schröder. Simulations and bisimulations for coalgebraic modal logics. In *Algebra and Coalgebra in Computer Science - 5th International Conference, CALCO 2013*, volume 8089 of *LNCS*, pages 253–266. Springer, 2013. `doi:10.1007/978-3-642-40206-7_19`.

25 H. Peter Gumm and Tobias Schröder. Monoid-labeled transition systems. In *Coalgebraic Methods in Computer Science, CMCS 2001*, volume 44(1) of *ENTCS*, pages 185–204. Elsevier, 2001. `doi:10.1016/S1571-0661(04)80908-3`.

26 H.Peter Gumm. From T-coalgebras to filter structures and transition systems. In *Algebra and Coalgebra in Computer Science*, volume 3629 of *LNCS*, pages 194–212. Springer, 2005.

27 John Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. In *Theory of Machines and Computations*, pages 189–196. Academic Press, 1971.

28 Paris C. Kanellakis and Scott A. Smolka. Ccs expressions, finite state processes, and three problems of equivalence. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, pages 228–240. ACM, 1983.

29 Paris C. Kanellakis and Scott A. Smolka. CCS expressions, finite state processes, and three problems of equivalence. *Inf. Comput.*, 86(1):43–68, 1990. `doi:10.1016/0890-5401(90)90025-D`.

30 Bartek Klin. The least fibred lifting and the expressivity of coalgebraic modal logic. In *Algebra and Coalgebra in Computer Science, CALCO 2005*, volume 3629 of *LNCS*, pages 247–262. Springer, 2005. `doi:10.1007/11548133_16`.

31 Timo Knuutila. Re-describing an algorithm by Hopcroft. *Theor. Comput. Sci.*, 250:333–363, 2001.

32 Barbara König, Christina Mika-Michalski, and Lutz Schröder. Explaining non-bisimilarity in a coalgebraic approach: Games and distinguishing formulas. In *Coalgebraic Methods in Computer Science*, pages 133–154. Springer, 2020.

33 Kim Guldstrand Larsen and Arne Arne Skou. Bisimulation through probabilistic testing. *Inform. Comput.*, 94(1):1–28, 1991. doi:10.1016/0890-5401(91)90030-6.

34 Johannes Marti and Yde Venema. Lax extensions of coalgebra functors and their logic. *J. Comput. Syst. Sci.*, 81(5):880–900, 2015. doi:10.1016/j.jcss.2014.12.006.

35 R. Milner. *Communication and Concurrency*. International series in computer science. Prentice-Hall, 1989.

36 Robert Paige and Robert E. Tarjan. Three partition refinement algorithms. *SIAM J. Comput.*, 16(6):973–989, 1987.

37 D. Park. Concurrency and automata on infinite sequences. In *Proceedings of 5th GI-Conference on Theoretical Computer Science*, volume 104 of *LNCS*, pages 167–183, 1981.

38 Dirk Pattinson. Coalgebraic modal logic: soundness, completeness and decidability of local consequence. *Theoretical Computer Science*, 309(1):177–193, 2003.

39 Dirk Pattinson. Expressive logics for coalgebras via terminal sequence induction. *Notre Dame J. Formal Log.*, 45(1):19–33, 2004. doi:10.1305/ndjfl/1094155277.

40 Jan Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249:3–80, 2000.

41 Jan Rutten and Erik de Vink. Bisimulation for probabilistic transition systems: a coalgebraic approach. *Theoret. Comput. Sci.*, 221:271–293, 1999.

42 Lutz Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. *Theor. Comput. Sci.*, 390(2-3):230–247, 2008. doi:10.1016/j.tcs.2007.09.023.

43 Věra Trnková. On a descriptive classification of set functors I. *Commentationes Mathematicae Universitatis Carolinae*, 12(1):143–174, 1971.

44 Antti Valmari and Giuliana Franceschinis. Simple $\mathcal{O}(m \log n)$ time Markov chain lumping. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2010*, volume 6015 of *LNCS*, pages 38–52. Springer, 2010.

45 Antti Valmari and Petri Lehtinen. Efficient minimization of dfas with partial transition. In *Theoretical Aspects of Computer Science, STACS 2008*, volume 1 of *LIPIcs*, pages 645–656. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2008. doi:10.4230/LIPIcs.STACS.2008.1328.

46 Thorsten Wißmann, Ulrich Dorsch, Stefan Milius, and Lutz Schröder. Efficient and Modular Coalgebraic Partition Refinement. *Logical Methods in Computer Science*, Volume 16, Issue 1, January 2020.

Enabling Preserving Bisimulation Equivalence

Rob van Glabbeek
Data61, CSIRO, Sydney, Australia
Computer Science and Engineering, University of New South Wales, Sydney, Australia

Peter Höfner
School of Computing, Australian National University, Canberra, Australia
Data61, CSIRO, Sydney, Australia

Weiyou Wang
School of Computing, Australian National University, Canberra, Australia

Abstract

Most fairness assumptions used for verifying liveness properties are criticised for being too strong or unrealistic. On the other hand, justness, arguably the minimal fairness assumption required for the verification of liveness properties, is not preserved by classical semantic equivalences, such as strong bisimilarity. To overcome this deficiency, we introduce a finer alternative to strong bisimilarity, called enabling preserving bisimilarity. We prove that this equivalence is justness-preserving and a congruence for all standard operators, including parallel composition.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Process calculi; Theory of computation → Distributed computing models

Keywords and phrases bisimilarity, liveness properties, fairness assumptions, process algebra

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.33

Related Version *Full Version*: `http://arxiv.org/abs/2108.00142` [15]

Acknowledgements We are grateful to Filippo De Bortoli.

1 Introduction

Formal verification of concurrent systems becomes more and more standard practice, in particular in safety-critical environments. Progress and fairness assumptions have to be used when verifying liveness properties, which guarantee that "something good will eventually happen". Without assumptions of this kind, no meaningful liveness property can formally be proven.

▶ **Example 1.** Consider the program `while(true) do x:=x+1 od` with `x` initialised to `0`. Intuitively, any liveness property of the form "eventually `x=n`" should be satisfied by the program. However, these properties are valid only when assuming *progress*, stating that a system will make progress when it can; otherwise the program could just stop after some computation. ⌟

Progress itself is not a strong enough assumption when concurrent systems are verified, for a system of multiple completely independent components makes progress as long as one of its components makes progress, even when others do not. For decades, researchers have developed notions of fairness and used them in both system specification and verification; the most common ones are surveyed in [13]. Two of the most popular fairness assumptions are weak and strong fairness of instructions [5].[1] They apply to systems whose behaviour is specified by some kind of code, composed out of *instructions*. A *task* is any activity of the system that stems from a particular instruction; it is *enabled* when the system is ready to do

[1] Often these notions are referred to as weak and strong fairness without mentioning instructions; here, we follow the terminology of [13], which is more precise.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 33; pp. 33:1–33:20
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

some part of that task, and *executed* when the system performs some part of it. Now weak and strong fairness of instructions state that whenever a task is enabled persistently (for weak fairness) or infinitely often (for strong fairness), then it will be executed by the system. These fairness assumptions, as well as all others surveyed in [13],[2] imply progress.

Despite being commonly used, it has been argued that most fairness assumptions, including weak and strong fairness of instructions, are often too strong or unrealistic, "in the sense that run-of-the-mill implementations tend not to be fair" [13].

(Reactive) systems are often described by *labelled transition systems*, which model all activities as transitions going from state to state, labelled with *actions*. Some actions require synchronisation of the modelled system with its environment; they can occur only when both the system and the environment are ready to engage in this action. Such actions, and the transitions labelled with them, are called *blocking*.

▶ **Example 2.** Assume that every morning Alice has a choice between a slice of bread with jam or a bacon and egg roll. A corresponding transition system consists of one state with two transitions, each standing for one kind of breakfast. Both weak and strong fairness (of instructions) will force Alice to eventually have both types of breakfast, ruling out the possibility that Alice picks up jam every day as she is a vegetarian. ⌟

To address this issue, a weaker assumption, called justness, has been proposed. It has been formulated for reactive systems, modelled as labelled transition systems. *Justness* is the assumption that

> *Once a non-blocking transition is enabled that stems from a set of parallel components, one (or more) of these components will eventually partake in a transition. [13]*

Example 2 features only one component, Alice. Assuming justness, as expected, she now has the option to eat jam for the rest of her life. Let us now look at a more technical example.

▶ **Example 3.** We consider the following two programs, and assume that all variables are initialised by `0`.

```
while(true) do                        while(true) do  ||  x := 1;
  choose                                y := y+1;     ||
    if true then y := y+1;            od              ||
    if x = 0 then x := 1;
  end
od
```

The example on the left presents an infinite loop containing an internal nondeterministic choice. The conditional write `if x = 0 then x := 1` describes an atomic read-modify-write (RMW) operation[3]. Such operators, supported by modern hardware, read a memory location and simultaneously write a new value into it. This example is similar to Example 2 in the sense that the liveness property "eventually `x=1`" should not be satisfied as the program has a choice every time the loop body is executed.

The example on the right-hand side is similar, but the handling of variables `x` and `y` are managed by different components. As the two programs are independent from each other – they could be executed on different machines – the property "eventually `x=1`" should hold.

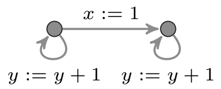
Justness differentiates these behaviour, whereas weak and strong fairness fail to do so. ⌟

[2] Many other notions of fairness are obtained by varying the definition of task. In *fairness of components* a task refers to all activity stemming from a component of a system that is a parallel composition.

[3] `https://en.wikipedia.org/wiki/Read-modify-write`

The above example illustrates that standard notions of fairness are regularly too strong, and the notion of justness may be a good replacement. When it comes to verification tasks, semantic equivalences, such as strong bisimilarity [18], are a standard tool to reduce the state space of the systems under consideration. Unfortunately, these semantic equivalences do not accord well with justness. The problem is that they are based on labelled transition systems, which do not capture the concept of components. The different behaviour of the two programs in Example 3 stems from the components involved. In fact, both programs give rise to the same transition system, depicted below. Systems featuring the same transition system cannot be distinguished by any semantic equivalence found in the literature. Consequently, the verification of the stated liveness property will fail for one of the two programs of Example 3.

To overcome this deficiency, we introduce *enabling preserving bisimilarity*, a finer alternative to strong bisimilarity, which respects justness. It is based on extended labelled transition systems that take components involved in particular transitions into account.



2 Labelled Transition Systems with Successors

As discussed in the introduction, one reason why strong bisimilarity does not preserve liveness properties under justness is that necessary information is missing, namely components.

The definition of *(parallel) components* was based on the parallel composition operator in process algebras when justness was first introduced in [4, 12], and has been generalised in later work to allow the use of justness in different contexts.

Here we define a justness-preserving semantic equivalence on an extension of labelled transition systems. Using labelled transition systems rather than process algebra as underlying concept makes our approach more general, for other models of concurrency, such as Petri nets or higher-dimensional automata, induce a semantics based on transition systems as well.

The essence of justness is that when a non-blocking transition t is enabled in a state s, eventually the system must perform an action u that *interferes* with it [13], notation $t \mathrel{\not\smile^{\bullet}} u$, in the sense that a component affected by u is necessary for the execution of t – or, to be more precise, for the variant t' of t that is enabled after the system has executed some actions that do not interfere with t. The present paper abstracts from the notion of component, but formalises justness, as well as our enabling preserving bisimilarity, in terms of a *successor relation* $t \leadsto_\pi t'$, marking t' as a successor of t, parametrised with the noninterfering actions π happening in between. This relation also encodes the above relation $\not\smile^{\bullet}$. The advantage of this approach over one that uses components explicitly, is that it also applies to models like higher-dimensional automata [21, 6, 16, 8] in which the notion of a component is more fluid, and changes during execution.

A *labelled transition system* (LTS) is a tuple $(S, \mathit{Tr}, \mathit{source}, \mathit{target}, \ell)$ with S and Tr sets (of *states* and *transitions*), $\mathit{source}, \mathit{target} : \mathit{Tr} \to S$ and $\ell : \mathit{Tr} \to \mathscr{L}$, for some set $\mathscr{L}$ of transition labels. A transition $t \in \mathit{Tr}$ of an LTS is *enabled* in a state $p \in S$ if $\mathit{source}(t) = p$. Let $\mathit{en}(p)$ be the set of transitions that are enabled in p.

A *path* in an LTS is an alternating sequence $p_0\, u_0\, p_1\, u_1\, p_2 \ldots$ of states and transitions, starting with a state and either being infinite or ending with a state, such that $\mathit{source}(u_i) = p_i$ and $\mathit{target}(u_i) = p_{i+1}$ for all relevant i. The *length* $l(\pi) \in \mathbb{N} \cup \{\infty\}$ of a path π is the number of transitions in it. If π is a path, then $\hat{\pi}$ is the sequence of transitions occurring in π.

▶ **Definition 4** (LTSS). A *labelled transition system with successors* (LTSS) is a tuple $(S, \mathit{Tr}, \mathit{source}, \mathit{target}, \ell, \leadsto)$ with $(S, \mathit{Tr}, \mathit{source}, \mathit{target}, \ell)$ an LTS, and $\leadsto \subseteq \mathit{Tr} \times \mathit{Tr} \times \mathit{Tr}$, the *successor relation*, such that if $(t, u, v) \in \leadsto$ then $\mathit{source}(t) = \mathit{source}(u)$ and $\mathit{source}(v) = \mathit{target}(u)$. We write $t \leadsto_u v$ for $(t, u, v) \in \leadsto$.

We use this successor relation to define the concept of (un)affected transitions. Let two transitions t and u be enabled in a state p, i.e., $t, u \in \mathit{en}(p)$ for some $p \in S$; the *concurrency relation* $\smile^{\bullet}$ is defined as $t \smile^{\bullet} u :\Leftrightarrow \exists v.\ t \leadsto_u v$. Its negation $t \not\smile^{\bullet} u$ says that the possible occurrence of t is *affected* by the occurrence of u. In case t is unaffected by u (i.e., $t \smile^{\bullet} u$), each v with $t \leadsto_u v$ denotes a variant of t that can occur after u. Note that the concurrency relation can be asymmetric. Examples are traffic lights – a car passing traffic lights should be affected by them, but the lights do not care whether the car is there; and read-write operations – reading shared memory can be affected by a write action, but, depending on how the memory is implemented, the opposite might not hold. In case t and u are mutually unaffected we write $t \smile u$, i.e., $t \smile u :\Leftrightarrow t \smile^{\bullet} u \wedge u \smile^{\bullet} t$.

It is possible to have $t \smile^{\bullet} t$, namely when executing transition t does not disable (a future variant of) t to occur again. This can happen when t is a signal emission, say of a traffic light shining red, for even after shining for some time it keeps on shining; or when t is a broadcast receive action, for receiving a broadcast does not invalidate a system's perpetual readiness to again accept a broadcast, either by receiving or ignoring it.

▶ **Example 5.** Consider the labelled transition system of Example 3, let t_1 and t_2 be the two transitions corresponding to `y:=y+1` in the first and second state, respectively, and let u be the transition for assignment `x:=1`. The assignments of `x` and `y` in the right-hand program are independent, hence $t_1 \leadsto_u t_2$, $u \leadsto_{t_1} u$ and $t_1 \smile u$.

For the program on the left-hand side, the situation is different. As the instructions stem from the same component (program), all transitions affect each other, i.e., $\leadsto\, = \,\smile^{\bullet}\, = \emptyset$. ⌟

The successor relation relates transitions one step apart. We lift it to sequences of transitions.

▶ **Definition 6** (Successor along Path). The relation $\leadsto$ is extended to $\leadsto\, \subseteq \mathit{Tr} \times \mathit{Tr}^* \times \mathit{Tr}$ by (i) $t \leadsto_\varepsilon w$ iff $w = t$, and (ii) $t \leadsto_{\pi u} w$ iff there is a v with $t \leadsto_\pi v$ and $v \leadsto_u w$.

Here, ε denotes the empty sequence, and πu the sequence π followed by transition u. We define a concurrency relation $\smile^{\bullet}\, \subseteq \mathit{Tr} \times \mathit{Tr}^*$ considering sequences of transitions by $t \smile^{\bullet} \pi :\Leftrightarrow \exists v.\ t \leadsto_\pi v$. Intuitively, $t \smile^{\bullet} \pi$ means that there exists a successor of t after π has been executed. Thus, t is unaffected by all transitions of π.

We are ready to define justness, which is parametrised by a set B of blocking actions.

▶ **Definition 7** (Justness). Given an LTSS $= (S, \mathit{Tr}, \mathit{source}, \mathit{target}, \ell, \leadsto)$ labelled over $\mathscr{L}$, and $B \subseteq \mathscr{L}$, a path π in is B-*just* if for each suffix π_0 of π and for each transition $t \in \mathit{Tr}^{\bullet}$ with $\ell(t) \notin B$ and $\mathit{source}(t)$ the first state of π_0, the path π_0 has a finite prefix ρ such that $t \not\smile^{\bullet} \hat{\rho}$. Here $\mathit{Tr}^{\bullet} := \{t \in \mathit{Tr} \mid t \not\smile^{\bullet} t\}$.

3 Enabling Preserving Bisimulation Equivalence

In this section we introduce enabling preserving bisimulation equivalence, and show how it preserves justness. In contrast to classical bisimulations, which are relations of type $S \times S$, the new equivalence is based on triples. The essence of justness is that a transition t enabled in a state s must eventually be affected by the sequence π of transitions the system performs. As long as π does not interfere with t, we obtain a transition t' with $t \leadsto_\pi t'$. This transition

t' represents the interests of t, and must eventually be affected by an extension of π. Here, executing t or t' as part of this extension is a valid way of interfering. To create a bisimulation that respects such considerations, for each related pair of states p and q we also match each enabled transition of p with one of q, and vice versa. These relations are maintained during the evolution of the two related systems, so that when one system finally interferes with a descendant of t, the related system interferes with the related descendant.

▶ **Definition 8** (Ep-bisimilarity)**.** Given an LTSS $(S, \mathit{Tr}, \mathit{source}, \mathit{target}, \ell, \leadsto)$, an *enabling preserving bisimulation* (*ep-bisimulation*) is a relation $\mathscr{R} \subseteq S \times S \times \mathscr{P}(\mathit{Tr} \times \mathit{Tr})$ satisfying

1. if $(p, q, R) \in \mathscr{R}$ then $R \subseteq \mathit{en}(p) \times \mathit{en}(q)$ such that
  a. $\forall t \in \mathit{en}(p).\ \exists\, u \in \mathit{en}(q).\ t\ R\ u$,
  b. $\forall u \in \mathit{en}(q).\ \exists\, t \in \mathit{en}(p).\ t\ R\ u$,
  c. if $t\ R\ u$ then $\ell(t) = \ell(u)$, and
2. if $(p, q, R) \in \mathscr{R}$ and $v\ R\ w$, then $(\mathit{target}(v), \mathit{target}(w), R') \in \mathscr{R}$ for some R' such that
  a. if $t\ R\ u$ and $t \leadsto_v t'$, then $\exists\, u'.\ u \leadsto_w u' \wedge t'\ R'\ u'$, and
  b. if $t\ R\ u$ and $u \leadsto_w u'$, then $\exists\, t'.\ t \leadsto_v t' \wedge t'\ R'\ u'$.

Two states p and q in an LTSS are *enabling preserving bisimilar* (ep-bisimilar), $p \mathrel{\underline{\leftrightarrow}}_{ep} q$, if there exists an enabling preserving bisimulation $\mathscr{R}$ such that $(p, q, R) \in \mathscr{R}$ for some R.

Definition 8 without Items 2a and 2b is nothing else than a reformulation of the classical definition of strong bisimilarity. An ep-bisimulation additionally maintains for each pair of related states p and q a relation R between the transitions enabled in p and q. Items 2a and 2b strengthen the condition on related target states by requiring that the successors of related transitions are again related relative to these target states. It is this requirement (and in particular its implication stated in the following observation) which distinguishes the transition systems for Example 3.

▶ **Observation 9.** Ep-bisimilarity respects the concurrency relation.
For a given ep-bisimulation $\mathscr{R}$, if $(p, q, R) \in \mathscr{R}$, $t\ R\ u$ and $v\ R\ w$ then $t \mathrel{\smile\!\!\!\bullet} v$ iff $u \mathrel{\smile\!\!\!\bullet} w$.

▶ **Proposition 10.** $\underline{\leftrightarrow}_{ep}$ is an equivalence relation.

Proof. *Reflexivity*: Let $(S, \mathit{Tr}, \mathit{source}, \mathit{target}, \ell, \leadsto)$ be an LTSS. The relation

$$\mathscr{R}_{\mathit{Id}} := \{(s, s, \mathit{Id}_s) \mid s \in S\}$$

is an ep-bisimulation. Here $\mathit{Id}_s := \{(t, t) \mid t \in \mathit{en}(s)\}$.
Symmetry: For a given ep-bisimulation $\mathscr{R}$, the relation

$$\mathscr{R}^{-1} := \{(q, p, R^{-1}) \mid (p, q, R) \in \mathscr{R}\}$$

is also an ep-bisimulation. Here $R^{-1} := \{(u, t) \mid (t, u) \in R\}$.
Transitivity: For given ep-bisimulations $\mathscr{R}_1$ and $\mathscr{R}_2$, the relation

$$\mathscr{R}_1; \mathscr{R}_2 := \{(p, r, R_1; R_2) \mid \exists\, q.\ (p, q, R_1) \in \mathscr{R}_1 \wedge (q, r, R_2) \in \mathscr{R}_2\}$$

is also an ep-bisimulation. Here $R_1; R_2 := \{(t, v) \mid \exists\, u.\ (t, u) \in R_1 \wedge (u, v) \in R_2\}$. ◀

▶ **Observation 11.** The union of any collection of ep-bisimulations is itself an ep-bisimulation.

Consequently there exists a largest ep-bisimulation.

Before proving that ep-bisimilarity preserves justness, we lift this relation to paths.

▶ **Definition 12** (Ep-bisimilarity of Paths)**.** Given an ep-bisimulation $\mathscr{R}$ and two paths $\pi = p_0\, u_0\, p_1\, u_1\, p_2 \dots$ and $\pi' = p'_0\, u'_0\, p'_1\, u'_1\, p'_2 \dots$, we write $\pi \mathrel{\mathscr{R}} \pi'$ iff $l(\pi) = l(\pi')$, and there exists $R_i \subseteq \mathit{Tr} \times \mathit{Tr}$ for all $i \in \mathbb{N}$ with $i \leq l(\pi)$, such that

1. $(p_i, p'_i, R_i) \in \mathscr{R}$ for each $i \in \mathbb{N}$ with $i \leq l(\pi)$,
2. $u_i \mathrel{R_i} u'_i$ for each $i < l(\pi)$,
3. if $t \mathrel{R_i} t'$ and $t \leadsto_{u_i} v$ with $i < l(\pi)$, then $\exists\, v'.\ t' \leadsto_{u'_i} v' \wedge v \mathrel{R_{i+1}} v'$, and
4. if $t \mathrel{R_i} t'$ and $t' \leadsto_{u'_i} v'$ with $i < l(\pi)$, then $\exists\, v.\ t \leadsto_{u_i} v \wedge v \mathrel{R_{i+1}} v'$.

Paths π and π' are *enabling preserving bisimilar*, notation $\pi \mathrel{\underline{\leftrightarrow}_{ep}} \pi'$, if there exists an ep-bisimulation $\mathscr{R}$ with $\pi \mathrel{\mathscr{R}} \pi'$. If $\pi \mathrel{\underline{\leftrightarrow}_{ep}} \pi'$, we also write $\pi \mathrel{\vec{R}} \pi'$ if $\vec{R} := (R_0, R_1, \dots)$ are the R_i required above.

Note that if $p \mathrel{\underline{\leftrightarrow}_{ep}} q$ and π is any path starting from p, then, by Definition 8, there is a path π' starting from q with $\pi \mathrel{\underline{\leftrightarrow}_{ep}} \pi'$. The following lemma lifts Observation 9.

▶ **Lemma 13.** If $\pi \mathrel{\vec{R}} \rho$ with π finite and $t \mathrel{R_0} t'$ then $t \mathrel{\smile^{\bullet}} \hat{\pi}$ iff $t' \mathrel{\smile^{\bullet}} \hat{\rho}$.

Proof. We have to show that $\exists v.\ t \leadsto_{\hat{\pi}} v$ iff $\exists v'.\ t' \leadsto_{\hat{\rho}} v'$. Using symmetry, we may restrict attention to the "only if" direction. We prove a slightly stronger statement, namely that for every transition v with $t \leadsto_{\hat{\pi}} v$ there exists a v' such that $t' \leadsto_{\hat{\rho}} v'$ and $v \mathrel{R_{l(\pi)}} v'$.

We proceed by induction on the length of π.

The base case, where $l(\pi) = 0$, $\hat{\pi} = \varepsilon$ and thus $v = t$, holds trivially, taking $v' := t'$.

So assume $\pi u p \mathrel{\vec{R}} \rho u' p'$ and $t \leadsto_{\hat{\pi} u} w$. Then there is a v with $t \leadsto_{\hat{\pi}} v$ and $v \leadsto_{u} w$. By induction there is a transition v' such that $t' \leadsto_{\hat{\rho}} v'$ and $v \mathrel{R_{l(\pi)}} v'$. By Definition 12(3), there is a w' with $v' \leadsto_{u'} w'$ and $w \mathrel{R_{l(\pi)+1}} w'$. Thus $t' \leadsto_{\hat{\rho} u'} w'$ by Definition 6. ◀

▶ **Theorem 14.** Ep-bisimilarity preserves justness: Given two paths π and π' in an LTSS with $\pi \mathrel{\underline{\leftrightarrow}_{ep}} \pi'$, and a set B of blocking actions, then π is B-just iff π' is B-just.

Proof. Let $\pi = p_0\, u_0\, p_1\, u_1\, p_2 \dots$ and $\pi' = p'_0\, u'_0\, p'_1\, u'_1\, p'_2 \dots$. Suppose π is B-unjust, so there exist an $i \in \mathbb{N}$ with $i \leq l(\pi)$ and a transition $t \in \mathit{Tr}^{\bullet}$ with $\ell(t) \notin B$ and $\mathit{source}(t) = p_i$ such that $t \mathrel{\smile^{\bullet}} \hat{\rho}$ for each finite prefix ρ of the suffix $p_i\, u_i\, p_{i+1}\, u_{i+1}\, p_{i+2} \dots$ of π. It suffices to show that also π' is B-unjust.

Take an ep-bisimulation $\mathscr{R}$ such that $\pi \mathrel{\mathscr{R}} \pi'$. Choose $R_i \subseteq \mathit{Tr} \times \mathit{Tr}$ for all $i \in \mathbb{N}$ with $i \leq l(\pi)$, satisfying the four conditions of Definition 12. Pick any $t' \in \mathit{Tr}$ with $t \mathrel{R_i} t'$ – such a t' must exist by Definition 8. Now $\mathit{source}(t') = p'_i$. By Definition 8, $t' \in \mathit{Tr}^{\bullet}$ and $\ell(t') = \ell(t) \notin B$. It remains to show that $t' \mathrel{\smile^{\bullet}} \hat{\rho}'$ for each finite prefix ρ' of the suffix $p'_i\, u'_i\, p'_{i+1}\, u'_{i+1}\, p'_{i+2} \dots$ of π'. This follows by Lemma 13. ◀

4 Stating and Verifying Liveness Properties

The main purpose of ep-bisimilarity is as a vehicle for proving liveness properties. A *liveness property* is any property saying that eventually something good will happen [17]. Liveness properties are *linear-time properties*, in the sense that they are interpreted primarily on the (complete) *runs* of a system. When a distributed system is formalised as a state in an (extended) LTS, a run of the distributed system is modelled as a path in the transition system, starting from that state. However, not every such path models a realistic system run. A *completeness criterion* [9] selects some of the paths of a system as *complete paths*, namely those that model runs of the represented system.

A state s in an (extended) LTS is said to satisfy a linear time property φ when employing the completeness criterion CC, notation $s \models^{CC} \varphi$, if each complete run of s satisfies φ [10]. Writing $\pi \models \varphi$ when property φ holds for path π, we thus have $s \models^{CC} \varphi$ iff $\pi \models \varphi$ for all complete paths π starting from s. When simplifying a system s into an equivalent system s', so that $s \sim s'$ for some equivalence relation $\sim$, it is important that judgements $\models^{CC} \varphi$ are preserved:

$$s \sim s' \quad \Rightarrow \quad (s \models^{CC} \varphi \Leftrightarrow s' \models^{CC} \varphi).$$

This is guaranteed when for each path π of s there exists a path π' of s', such that (a) $\pi \models \varphi$ iff $\pi' \models \varphi$, and (b) π' is complete iff π is complete. Here (a) is already guaranteed when π and π' are related by strong bisimilarity. Taking CC to be B-justness for any classification of a set of actions B as blocking, and $\sim$ to be $\underline{\leftrightarrow}_{ep}$, Theorem 14 now ensures (b) as well.

5 Interpreting Justness in Process Algebras

Rather than using LTSs directly to model distributed systems, one usually employs other formalisms such as process algebras or Petri nets, for they are often easier to use for system modelling. Their formal semantics maps their syntax into states of LTSs. In this section we introduce the *Algebra of Broadcast Communication with discards and Emissions* (ABCdE), an extension of Milner's *Calculus of Communication Systems* (CCS) [18] with broadcast communication and signal emissions. In particular, we give a structural operational semantics [19] that interprets process expressions as states in an LTS. Subsequently, we define the successor relation $\leadsto$ for ABCdE, thereby enriching the LTS into an LTSS.

We use ABCdE here, as for many realistic applications CCS is not expressive enough [7, 11]. The presented approach can be applied to a wide range of process algebras. ABCdE is largely designed to be a starting point for transferring the presented theory to algebras used for "real" applications. For example, broadcast communication is needed for verifying routing protocols (e.g. [14]); signals are employed to correctly render and verify protocols for mutual exclusion [11, 3]. Another reason is that broadcasts as well as signals, in different ways, give rise to asymmetric concurrency relations, and we want to show that our approach is flexible enough to handle this.

5.1 Algebra of Broadcast Communication with Discards and Emissions

ABCdE is parametrised with sets $\mathscr{A}$ of *agent identifiers*, $\mathscr{C}$ of *handshake communication names*, $\mathscr{B}$ of *broadcast communication names*, and $\mathscr{S}$ of *signals*; each $A \in \mathscr{A}$ comes with a defining equation $A \overset{def}{=} P$ with P being a guarded ABCdE expression as defined below. $\bar{\mathscr{C}} := \{\bar{c} \mid c \in \mathscr{C}\}$ is the set of *handshake communication co-names*, and $\bar{\mathscr{S}} := \{\bar{s} \mid s \in \mathscr{S}\}$ is the set of signal emissions. The collections $\mathscr{B}!$, $\mathscr{B}?$, and $\mathscr{B}{:}$ of *broadcast*, *receive*, and *discard* actions are given by $\mathscr{B}\sharp := \{b\sharp \mid b \in \mathscr{B}\}$ for $\sharp \in \{!, ?, :\}$. $\mathit{Act} := \mathscr{C} \uplus \bar{\mathscr{C}} \uplus \{\tau\} \uplus \mathscr{B}! \uplus \mathscr{B}? \uplus \mathscr{S}$ is the set of *actions*, where τ is a special *internal action*. $\mathscr{L} := \mathit{Act} \uplus \mathscr{B}{:} \uplus \bar{\mathscr{S}}$ is the set of *transition labels*. Complementation extends to $\mathscr{C} \uplus \bar{\mathscr{C}} \uplus \mathscr{S} \uplus \bar{\mathscr{S}}$ by $\bar{\bar{c}} := c$.

Below, c ranges over $\mathscr{C} \uplus \bar{\mathscr{C}} \uplus \mathscr{S} \uplus \bar{\mathscr{S}}$, η over $\mathscr{C} \uplus \bar{\mathscr{C}} \uplus \{\tau\} \uplus \mathscr{S} \uplus \bar{\mathscr{S}}$, α over Act, ℓ over $\mathscr{L}$, b over $\mathscr{B}$, $\sharp, \sharp_1, \sharp_2$ over $\{!, ?, :\}$ and s, r over $\mathscr{S}$. A *relabelling* is a function $f : (\mathscr{C} \to \mathscr{C}) \uplus (\mathscr{B} \to \mathscr{B}) \uplus (\mathscr{S} \to \mathscr{S})$; it extends to $\mathscr{L}$ by $f(\bar{c}) = \overline{f(c)}$, $f(\tau) := \tau$, and $f(b\sharp) = f(b)\sharp$.

The set $\mathbb{P}$ of ABCdE expressions or *processes* is the smallest set including:

$\mathbf{0}$		*inaction*	$\alpha.P$	for $\alpha \in \mathit{Act}$ and $P \in \mathbb{P}$	*action prefixing*
$P + Q$	for $P, Q \in \mathbb{P}$	*choice*	$P \mid Q$	for $P, Q \in \mathbb{P}$	*parallel composition*
$P \backslash L$	for $L \subseteq \mathscr{C} \uplus \mathscr{S}$, $P \in \mathbb{P}$	*restriction*	$P[f]$	for f a relabelling, $P \in \mathbb{P}$	*relabelling*
A	for $A \in \mathscr{A}$	*agent identifier*	$P\hat{\ }s$	for $s \in \mathscr{S}$	*signalling*

Table 1 Structural operational semantics of ABCdE – Basic.

$$\alpha.P \xrightarrow{\alpha} P \ (\textsc{Act}) \qquad \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \ (\textsc{Sum-l}) \qquad \frac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'} \ (\textsc{Sum-r})$$

$$\frac{P \xrightarrow{\eta} P'}{P|Q \xrightarrow{\eta} P'|Q} \ (\textsc{Par-l}) \qquad \frac{P \xrightarrow{c} P',\ Q \xrightarrow{\bar{c}} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \ (\textsc{Comm}) \qquad \frac{Q \xrightarrow{\eta} Q'}{P|Q \xrightarrow{\eta} P|Q'} \ (\textsc{Par-r})$$

$$\frac{P \xrightarrow{\ell} P'}{P\backslash L \xrightarrow{\ell} P'\backslash L} \ (\ell \notin L \cup \overline{L}) \ (\textsc{Res}) \qquad \frac{P \xrightarrow{\ell} P'}{P[f] \xrightarrow{f(\ell)} P'[f]} \ (\textsc{Rel}) \qquad \frac{P \xrightarrow{\alpha} P'}{A \xrightarrow{\alpha} P'} \ (A \overset{def}{=} P) \ (\textsc{Rec})$$

We abbreviate $\alpha.\mathbf{0}$ by α, and $P\backslash\{c\}$ by $P\backslash c$. An expression is guarded if each agent identifier occurs within the scope of a prefixing operator.

The semantics of ABCdE is given by the labelled transition relation $\rightarrow \subseteq \mathbb{P} \times \mathscr{L} \times \mathbb{P}$, where transitions $P \xrightarrow{\ell} Q$ are derived from the rules of Tables 1–3. Here $\overline{L} := \{\bar{c} \mid c \in L\}$.

Table 1 shows the basic operational semantics rules, identical to the ones of CCS [18]. The process $\alpha.P$ performs the action α first and subsequently acts as P. The choice operator $P + Q$ may act as either P or Q, depending on which of the processes is able to act at all. The parallel composition $P|Q$ executes an action η from P, an action η from Q, or in the case where P and Q can perform complementary actions c and $\bar{c}$, the process can perform a synchronisation, resulting in an internal action τ. The restriction operator $P\backslash L$ inhibits execution of the actions from L and their complements. The relabelling $P[f]$ acts like process P with all labels ℓ replaced by $f(\ell)$. Finally, an agent A can do the same actions as the body P of its defining equation. When we take $\mathscr{B} = \mathscr{S} := \emptyset$, only the rules of Table 1 matter, and ABCdE simplifies to CCS.

Table 2 augments CCS with a mechanism for broadcast communication. The rules are similar to the ones for the Calculus of Broadcasting Systems (CBS) [20]; they also appear in the process algebra ABC [12], a strict subalgebra of ABCdE. The Rule (Bro) presents the core of broadcast communication, where any broadcast-action $b!$ performed by a component in a parallel composition is guaranteed to be received by any other component that is ready to do so, i.e., in a state that admits a $b?$-transition. Since it is vital that the sender of a broadcast can always proceed with it, regardless of the state of other processes running in parallel, the process algebra features discard actions $b{:}$, in such a way that each process in any state can either receive a particular broadcast b, by performing the action $b?$, or discard it, by means of a $b{:}$, but not both. A broadcast transmission $b!$ can synchronise with either $b?$ or $b{:}$, and thus is never blocked by lack of a listening party. In order to ensure associativity of the parallel composition, one requires rule (Bro) to consider receipt at the same time ($\sharp_1 = \sharp_2 = ?$). The remaining four rules of Table 2 generate the discard-actions. The Rule (Dis-nil) allows the nil process (inaction) to discard any incoming message; in the same spirit (Dis-act) allows a

Table 2 Structural operational semantics of ABCdE – Broadcast.

$$\mathbf{0} \xrightarrow{b:} \mathbf{0} \ (\textsc{Dis-nil}) \qquad \alpha.P \xrightarrow{b:} \alpha.P \ (\alpha \neq b?) \ (\textsc{Dis-act}) \qquad \frac{P \xrightarrow{b:} P',\ Q \xrightarrow{b:} Q'}{P+Q \xrightarrow{b:} P'+Q'} \ (\textsc{Dis-sum})$$

$$\frac{P \xrightarrow{b\sharp_1} P',\ Q \xrightarrow{b\sharp_2} Q'}{P|Q \xrightarrow{b\sharp} P'|Q'} (\sharp_1 \circ \sharp_2 = \sharp \neq \_) \text{ with } \begin{array}{c|ccc} \circ & ! & ? & : \\ \hline ! & \_ & ! & ! \\ ? & ! & ? & ? \\ : & ! & ? & : \end{array} \ (\textsc{Bro}) \qquad \frac{P \xrightarrow{b:} P'}{A \xrightarrow{b:} A} \ (A \overset{def}{=} P) \ (\textsc{Dis-rec})$$

Table 3 Structural operational semantics of ABCdE – Signals.

$$P\,\hat{}\,s \xrightarrow{\bar{s}} P\,\hat{}\,s \ (\textsc{Sig}) \qquad \frac{P \xrightarrow{\bar{s}} P'}{P+Q \xrightarrow{\bar{s}} P'+Q}\ (\textsc{Sig-sum-l}) \qquad \frac{Q \xrightarrow{\bar{s}} Q'}{P+Q \xrightarrow{\bar{s}} P+Q'}\ (\textsc{Sig-sum-r})$$

$$\frac{P \xrightarrow{\bar{s}} P'}{P\,\hat{}\,r \xrightarrow{\bar{s}} P'\,\hat{}\,r}\ (\textsc{Sig-sig}) \qquad \frac{P \xrightarrow{\bar{s}} P'}{A \xrightarrow{\bar{s}} A}\ (A \overset{def}{=} P)\ (\textsc{Sig-rec}) \qquad \frac{P \xrightarrow{\alpha} P'}{P\,\hat{}\,r \xrightarrow{\alpha} P'}\ (\textsc{Act-sig}) \qquad \frac{P \xrightarrow{b:} P'}{P\,\hat{}\,r \xrightarrow{b:} P'\,\hat{}\,r}\ (\textsc{Dis-sig})$$

message to be discarded by a process that cannot receive it. A process offering a choice can only perform a discard-action if neither choice-option can handle it (Rule (Dis-sum)). Finally, an agent A can discard a broadcast iff the body P of its defining equation can discard it. Note that in all these cases a process does not change state by discarding a broadcast.

There exists a variant of CBS, ABC and ABCdE without discard actions, see [12, 9]. This approach, however, features negative premises in the operational rules. As a consequence, the semantics are not in De Simone format [23]. Making use of discard actions and staying within the De Simone format allows us to use meta-theory about this particular format. For example we know, without producing our own proof, that the operators $+$ and $|$ of ABC and ABCdE are associative and commutative, up to strong bisimilarity [2]. Moreover, strong bisimilarity [18] is a congruence for all operators of ABCdE.

Next to the standard operators of CCS and a broadcast mechanism, ABCdE features also signal emission. Informally, the signalling operator $P\,\hat{}\,s$ emits the signal s to be read by another process. Signal emissions cannot block other actions of P. Classical examples are the modelling of read-write processes or traffic lights (see Section 2).

Formally, our process algebra features a set $\mathscr{S}$ of signals. The semantics of signals is presented in Table 3. The first rule (Sig) models the emission $\bar{s}$ of signal s to the environment. The environment (processes running in parallel) can read the signal by performing a read action s. This action synchronises with the emission $\bar{s}$, via the rules of Table 1. Reading does not change the state of the emitter. The next four rules describe the interaction between signal emission and other operators, namely choice, signal emission and recursion. In short, these operators do not prevent the emission of a signal, and emitting signals never changes the state of the emitting process. Other operators, such as relabelling and restriction do not need special attention as they are already handled by the corresponding rules in Table 1. This is achieved by carefully selecting the types of the labels: while (Sum-l) features a label α of type Act, the rules for restriction and relabelling use a label $\ell \in \mathscr{L}$. In case a process performs a "proper" action, the signal emission ceases (Rule (Act-sig)), but if the process performs a broadcast discard transition, it does not (Rule (Dis-sig)).

The presented semantics stays within the realm of the De Simone format [23], which brings many advantages. However, there exists an alternative, equivalent semantics, which is based on predicates. Rather than explicitly modelling P emitting s by the transition $P \xrightarrow{\bar{s}} P$, one can introduce the predicate $P^{\curvearrowright s}$. The full semantics can be found in [3]. Some readers might find this notation more intuitive as signal emitting processes do not perform an actual action when a component reads the emitted signal.

5.2 Naming Transitions

The operational semantics of ABCdE presented in Section 5.1 interprets the language as an LTS. In Section 5.3, we aim to extend this LTS into an LTSS by defining a successor relation $\leadsto$ on the transitions, and thereby also a concurrency relation $\smile\!\!\bullet$. However, the standard interpretation of CCS-like languages, which takes as transitions the triples $P \xrightarrow{\alpha} Q$ that are derivable from the operational rules, does not work for our purpose.

▶ **Example 15.** Let $P = A|B$ with $A \overset{def}{=} \tau.A + a.A$ and $B \overset{def}{=} \bar{a}.B$. Now the transition $P \xrightarrow{\tau} P$ arises in two ways; either as a transition solely of the left component, or as a synchronisation between both components. The first form of that transition is concurrent with the transition $P \xrightarrow{\bar{a}} P$, but the second is not. In fact, an infinite path that would only perform the τ-transition stemming from the left component would not be just, whereas a path that schedules both τ-transitions infinitely often is. This shows that we want to distinguish these two τ-transitions, and hence not see a transition as a triple $P \xrightarrow{\alpha} Q$. ⌟

Instead, we take as the set *Tr* of transitions in our LTSS the *derivations* of the *transition triples* $P \xrightarrow{\alpha} Q$ from our operational rules. This is our reason to start with a definition of an LTS that features transitions as a primitive rather than a derived concept.

▶ **Definition 16** (Derivation). A *derivation* of a transition triple φ is a *well-founded* (without infinite paths that keep going up), ordered, upwardly branching tree with the nodes labelled by transition triples, such that

1. the root is labelled by φ, and
2. if μ is the label of a node and K is the sequence of labels of this node's children then $\frac{K}{\mu}$ is a substitution instance of a rule from Tables 1–3.

Given a derivation, we refer to the subtrees obtained by deleting the root node as its *direct subderivations.* Furthermore, by definition, $\frac{K_\varphi}{\varphi}$ is a substitution instance of a rule, where φ is the label of the derivation's root and K_φ is the sequence of labels of the root's children; the derivation is said to be *obtained* by this rule.

We interpret ABCdE as an LTS $(S, \mathit{Tr}, \mathit{source}, \mathit{target}, \ell)$ by taking as states S the ABCdE expressions $\mathbb{P}$ and as transitions *Tr* the derivations of transition triples $P \xrightarrow{\alpha} Q$. Given a derivation t of a triple $P \xrightarrow{\alpha} Q$, we define its label $\ell(t) \coloneqq \alpha$, its source $\mathit{source}(t) \coloneqq P$, and its target $\mathit{target}(t) \coloneqq Q$.

▶ **Definition 17** (Name of Derivation). The derivation obtained by application of (Act) is called $\xrightarrow{\alpha}P$. The derivation obtained by application of (Comm) or (Bro) is called $t|u$, where t, u are the names of its direct subderivations.[4] The derivation obtained by application of (Par-l) is called $t|Q$ where t is the direct subderivation's name and Q is the process on the right hand side of $|$ in the derivation's source. In the same way, the derivation obtained by application of (Par-r) is called $P|t$, while (Sum-l), (Sum-r), (Res), (Rel), and (Rec) yield $t{+}Q$, $P{+}t$, $t\backslash L$, $t[f]$ and $A{:}t$, where t is the direct subderivation's name. The remaining four rules of Table 2 yield $b{:}\mathbf{0}$, $b{:}\alpha.P$, $t{+}u$ and $A{:}t$, where t, u are direct subderivations' names. The derivation of $P\hat{}s \xrightarrow{\bar{s}} P\hat{}s$ obtained by (Sig) is called $P^{\to s}$. Rules (Act-sig), (Dis-sig) and (Sig-sig) yield $t\hat{}r$, and rules (Sig-sum-l), (Sig-sum-r) and (Sig-rec) yield $t{+}Q$, $P{+}t$ and $A{:}t$, where t is the direct subderivation's name.

A derivation's name reflects the syntactic structure of that derivation. The derivations' names not only provide a convenient way to identify derivations but also highlight the compositionality of derivations. For example, given a derivation t of $P \xrightarrow{c} P'$ and a derivation u of $Q \xrightarrow{\bar{c}} Q'$ with $c \in \mathscr{C} \mathbin{\dot\cup} \bar{\mathscr{C}} \mathbin{\dot\cup} \mathscr{S} \mathbin{\dot\cup} \bar{\mathscr{S}}$, $t|u$ will be a derivation of $P|Q \xrightarrow{\tau} P'|Q'$.

Hereafter, we refer to a derivation of a transition triple as a "transition".

[4] The order of a rule's premises should be maintained in the names of derivations obtained by it. Here t should be the derivation corresponding to the first premise and u to the second. As a result, $t \neq u \implies t|u \neq u|t$.

5.3 Successors

In this section we extend the LTS of ABCdE into an LTSS, by defining the successor relation $\rightsquigarrow$. For didactic reasons, we do so first for CCS, and then extend our work to ABCdE.

Note that $\chi \rightsquigarrow_\zeta \chi'$ can hold only when $\mathit{source}(\chi) = \mathit{source}(\zeta)$, i.e., transitions χ and ζ are both enabled in the state $O := \mathit{source}(\chi) = \mathit{source}(\zeta)$. It can thus be defined by structural induction on O. The meaning of $\chi \rightsquigarrow_\zeta \chi'$ is (a) that χ is unaffected by ζ – denoted $\chi \mathrel{\smile^{\bullet}} \zeta$ – and (b) that when doing ζ instead of χ, afterwards a variant χ' of χ is still enabled. Restricted to CCS, the relation $\mathrel{\smile^{\bullet}}$ is moreover symmetric, and we can write $\chi \smile \zeta$.

In the special case that $O = \mathbf{0}$ or $O = \alpha.Q$, there are no two concurrent transitions enabled in O, so this yields no triples $\chi \rightsquigarrow_\zeta \chi'$. When $O = P + Q$, any two concurrent transitions $\chi \smile \zeta$ enabled in O must either stem both from P or both from Q. In the former case, these transitions have the form $\chi = t + Q$ and $\zeta = v + Q$, and we must have $t \smile v$, in the sense that t and v stem from different parallel components within P. So $t \rightsquigarrow_v t'$ for some transition t'. As the execution of ζ discards the summand Q, we also obtain $\chi \rightsquigarrow_\zeta t'$. This motivates Item 1 in Definition 18 below. Item 2 follows by symmetry.

Let $O = P|Q$. One possibility for $\chi \rightsquigarrow_\zeta \chi'$ is that χ comes from the left component and ζ from the right. So χ has the form $t|Q$ and $\zeta = P|w$. In that case χ and ζ must be concurrent: we always have $\chi \smile \zeta$. When doing w on the right, the left component does not change, and afterwards t is still possible. Hence $\chi \rightsquigarrow_\zeta t|\mathit{target}(w)$. This explains Item 3 in Definition 18.

Another possibility is that χ and ζ both stem from the left component. In that case $\chi = t|Q$ and $\zeta = v|Q$, and it must be that $t \smile u$ within the left component. Thus $t \rightsquigarrow_v t'$ for some transition t', and we obtain $\chi \rightsquigarrow_\zeta t'|Q$. This motivates the first part of Item 4.

It can also happen that χ stems form the left component, whereas ζ is a synchronisation, involving both components. Thus $\chi = t|Q$ and $\zeta = v|w$. For $\chi \smile \zeta$ to hold, it must be that $t \smile v$, whereas the w-part of ζ cannot interfere with t. This yields the second part of Item 4.

The last part of Item 4 is explained in a similar vain from the possibility that ζ stems from the left while χ is a synchronisation of both components. Item 5 follows by symmetry.

In case both χ and ζ are synchronisations involving both components, i.e., $\chi = t|u$ and $\zeta = v|w$, it must be that $t \smile v$ and $u \smile w$. Now the resulting variant χ' of χ after ζ is simply $t'|v'$, where $t \rightsquigarrow_v t'$ and $u \rightsquigarrow_v u'$. This underpins Item 6.

If O has the form $P[f]$, χ and ζ must have the form $t[f]$ and $v[f]$, respectively. Whether t and v are concurrent is not influenced by the renaming operator. So $t \smile v$. The variant of t that remains after doing v is also not affected by the renaming, so if $t \rightsquigarrow_v t'$ then $\chi \rightsquigarrow_\zeta t'[f]$. The case that O has the form $P\backslash L$ is equally trivial. This yields the first two parts of Item 7.

In case $O = A$ with $A \overset{\mathit{def}}{=} P$, then χ and ζ must have the forms $A{:}t$ and $A{:}v$, respectively, where t and v are enabled in P. Now $\chi \smile \zeta$ only if $t \smile v$, so $t \rightsquigarrow_v t'$ for some transition t'. As the recursion around P disappears upon executing ζ, we obtain $\chi \rightsquigarrow_\zeta t'$. This yields the last part of Item 7. Together, this motivates the following definition.

▶ **Definition 18** (Successor Relation for CCS). The relation $\rightsquigarrow \subseteq \mathit{Tr} \times \mathit{Tr} \times \mathit{Tr}$ is the smallest relation satisfying

1. $t \rightsquigarrow_v t'$ implies $t + Q \rightsquigarrow_{v+Q} t'$,
2. $u \rightsquigarrow_w u'$ implies $P + u \rightsquigarrow_{P+w} u'$,
3. $t|Q \rightsquigarrow_{P|w} (t|\mathit{target}(w))$ and $P|u \rightsquigarrow_{v|Q} (\mathit{target}(v)|u)$,
4. $t \rightsquigarrow_v t'$ implies $t|Q \rightsquigarrow_{v|Q} t'|Q$, $t|Q \rightsquigarrow_{v|w} (t'|\mathit{target}(w))$, and $t|u \rightsquigarrow_{v|Q} t'|u$,
5. $u \rightsquigarrow_w u'$ implies $P|u \rightsquigarrow_{P|w} P|u'$, $P|u \rightsquigarrow_{v|w} (\mathit{target}(v)|u')$, and $t|u \rightsquigarrow_{P|w} t|u'$,
6. $t \rightsquigarrow_v t' \wedge u \rightsquigarrow_w u'$ implies $t|u \rightsquigarrow_{v|w} t'|u'$,
7. $t \rightsquigarrow_v t'$ implies $t\backslash L \rightsquigarrow_{v\backslash L} t'\backslash L$, $t[f] \rightsquigarrow_{v[f]} t'[f]$ and $A{:}t \rightsquigarrow_{A:v} t'$.

for all $t, t', u, u', v, w \in \mathit{Tr}$, $P, Q \in \mathbb{P}$ and L, f, A with $\mathit{source}(t) = \mathit{source}(v) = P$, $\mathit{source}(u) = \mathit{source}(w) = Q$, $\mathit{source}(t') = \mathit{target}(v)$, $\mathit{source}(u') = \mathit{target}(w)$, $L \subseteq \mathscr{C}$, f a relabelling and $A \in \mathscr{A}$ – provided that the composed transitions exist.

By projecting the ternary relation $\leadsto$ on its first two components, we obtain a characterisation of the concurrency relation $\smile$ between CCS transitions:

▶ **Observation 19** (Concurrency Relation for CCS). The relation $\smile \subseteq \mathit{Tr} \times \mathit{Tr}$ is the smallest relation satisfying

1. $t \smile v$ implies $t + Q \smile v + Q$,
2. $u \smile w$ implies $P + u \smile P + w$,
3. $t|Q \smile P|w$ and $P|u \smile v|Q$,
4. $t \smile v$ implies $t|Q \smile v|Q$, $t|Q \smile v|w$, and $t|u \smile v|Q$,
5. $u \smile w$ implies $P|u \smile P|w$, $P|u \smile v|w$, and $t|u \smile P|w$,
6. $t \smile v \wedge u \smile w$ implies $t|u \smile v|w$,
7. $t \smile v$ implies $t\backslash L \smile v\backslash L$, $t[f] \smile v[f]$ and $A{:}t \smile A{:}v$,

for all $t, u, v, w \in \mathit{Tr}$, $P, Q \in \mathbb{P}$ and L, f, A with $\mathit{source}(t) = \mathit{source}(v) = P$, $\mathit{source}(u) = \mathit{source}(w) = Q$, $L \subseteq \mathscr{C}$, f a relabelling and $A \in \mathscr{A}$ – provided that the composed transitions exist.

The same concurrency relation appeared earlier in [12]. Definition 18 and Observation 19 implicitly provide SOS rules for $\leadsto$ and $\smile$, such as $\frac{t \smile v'}{t+Q \smile v+Q}$. It is part of future work to investigate a rule format for ep-bisimilarity.

Definition 20 below generalises Definition 18 to all of ABCdE. In the special case that ζ is a broadcast discard or signal emission, i.e., $\ell(\zeta) \in \mathscr{B}{:} \cup \bar{\mathscr{S}}$, the transition ζ does not change state – we have $\mathit{source}(\zeta) = \mathit{target}(\zeta) = O$ – and is supposed not to interfere with any other transition χ enabled in O. Hence $\chi \mathrel{\smile^{\bullet}} \zeta$ and $\chi \leadsto_\zeta \chi$. This is Item 1 from Definition 20.

Consequently, in Item 11, which corresponds to Item 7 from Definition 18, we can now safely restrict attention to the case $\ell(\zeta) \in \mathit{Act}$. The last part of that item says that if within the scope of a signalling operator an action v occurs, one escapes from this signalling context, similarly to the cases of choice and recursion. That would not apply if v is a broadcast discard or signal emission, however.

An interesting case is when χ is a broadcast receive or discard transition, i.e., $\ell(\chi) = b?$ or $b{:}$. We postulate that one can never interfere with such an activity, as each process is always able to synchronise with a broadcast action, either by receiving or by discarding it. So we have $\chi \mathrel{\smile^{\bullet}} \zeta$ for all ζ with $O = \mathit{source}(\zeta) = \mathit{source}(\chi)$. It could be, however, that in $\chi \leadsto_\zeta \chi'$, one has $\ell(\chi) = b?$ and $\ell(\chi') = b{:}$, or vice versa. Item 2 says that if $O = \alpha.P$, with ζ the α-transition to P, then χ' can be any transition labelled $b?$ or $b{:}$ that is enabled in P. The second parts of Items 3 and 4 generalise this idea to discard actions enabled in a state of the form $P + Q$. Finally, Items 5 and 6 state that when χ is a broadcast receive stemming from the left side of $O = P + Q$ and ζ an action from the right, or vice versa, then χ' may be any transition labelled $b?$ or $b{:}$ that is enabled in $\mathit{target}(\zeta)$. In all other cases, successors of χ are inherited from successors of their building block, similar to the cases of other transitions.

▶ **Definition 20** (Successor Relation for ABCdE). The relation $\leadsto \subseteq \mathit{Tr} \times \mathit{Tr} \times \mathit{Tr}$ is the smallest relation satisfying

1. $\ell(\zeta) \in \mathscr{B}{:} \cup \bar{\mathscr{S}}$ and $\mathit{source}(\zeta) = \mathit{source}(\chi)$ implies $\chi \leadsto_\zeta \chi$,
2. $\ell(t) \in \{b?, b{:}\}$ implies $\overset{b?}{\to}P \leadsto_{\overset{b?}{\to}P} t$ and $b{:}\alpha.P \leadsto_{\overset{\alpha}{\to}P} t$,
3. $\ell(v) \notin \bar{\mathscr{S}} \wedge t \leadsto_v t'$ implies $t + Q \leadsto_{v+Q} t'$ and $t + u \leadsto_{v+Q} t'$,
4. $\ell(w) \notin \bar{\mathscr{S}} \wedge u \leadsto_w u'$ implies $P + u \leadsto_{P+w} u'$ and $t + u \leadsto_{P+w} u'$,
5. $\ell(w) \notin \bar{\mathscr{S}} \wedge \ell(t) = b? \wedge \ell(u') \in \{b?, b{:}\}$ implies $t + Q \leadsto_{P+w} u'$,

6. $\ell(v) \notin \bar{\mathscr{S}} \wedge \ell(u) = b? \wedge \ell(t') \in \{b?, b{:}\}$ implies $P + u \leadsto_{v+Q} t'$,
7. $t|Q \leadsto_{P|w} (t|\mathit{target}(w))$ and $P|u \leadsto_{v|Q} (\mathit{target}(v)|u)$,
8. $t \leadsto_v t'$ implies $t|Q \leadsto_{v|Q} t'|Q$, $t|Q \leadsto_{v|w} (t'|\mathit{target}(w))$, and $t|u \leadsto_{v|Q} t'|u$,
9. $u \leadsto_w u'$ implies $P|u \leadsto_{P|w} P|u'$, $P|u \leadsto_{v|w} (\mathit{target}(v)|u')$, and $t|u \leadsto_{P|w} t|u'$,
10. $t \leadsto_v t' \wedge u \leadsto_w u'$ implies $t|u \leadsto_{v|w} t'|u'$,
11. $\ell(v) \in \mathit{Act} \wedge t \leadsto_v t'$ implies $t\backslash L \leadsto_{v\backslash L} t'\backslash L$, $t[f] \leadsto_{v[f]} t'[f]$, $A{:}t \leadsto_{A:v} t'$ and $t\,\hat{}\,r \leadsto_{v\hat{}r} t'$,

for all $t, t', u, u', v, w \in \mathit{Tr}$, $P, Q \in \mathbb{P}$ and α, L, f, A, b, r with $\mathit{source}(t) = \mathit{source}(v) = P$, $\mathit{source}(u){=}\mathit{source}(w){=}Q$, $\mathit{source}(t'){=}\mathit{target}(v)$ and $\mathit{source}(u'){=}\mathit{target}(w)$, $\alpha \in \mathit{Act}$, $L \subseteq \mathscr{C} \dot\cup \mathscr{S}$, f a relabelling, $A \in \mathscr{A}$, $b \in \mathscr{B}$ and $r \in \mathscr{S}$ – provided that the composed transitions exist.

Although we have chosen to inductively define the $\leadsto$ relations, in [15, Appendix B] we follow a different approach in which Definition 20 appears as a theorem rather than a definition. Following [9], we understand each transition as the synchronisation of a number of elementary particles called *synchrons*. Then relations on synchrons are proposed in terms of which the $\leadsto$ relation is defined. That this leads to the same result indicates that the above definition is more principled than arbitrary.

5.4 Congruence and Other Basic Properties of Ep-bisimilarity

As mentioned before, the operators $+$ and $|$ are associative and commutative up to strong bisimilarity. We can strengthen this result.

▶ **Theorem 21.** *The operators $+$ and $|$ are associative and commutative up to $\mathrel{\underline{\leftrightarrow}}_{ep}$.*

Proof. Remember that $\mathbb{P}$ denotes the set of ABCdE expressions or processes. *Commutativity of $+$, i.e., $P + Q \mathrel{\underline{\leftrightarrow}}_{ep} Q + P$:* The relation

$$\{(I, I, \mathit{Id}_I) \mid I \in \mathbb{P}\} \dot\cup \{(P + Q, Q + P, R_{P,Q}) \mid P, Q \in \mathbb{P}\}$$

is an ep-bisimulation. Here $\mathit{Id}_I := \{(t, t) \mid t \in \mathit{en}(I)\}$ and

$$\begin{aligned} R_{P,Q} :=& \{(t + Q, Q + t) \mid t \in \mathit{en}(P) \wedge \ell(t) \notin \mathscr{B}{:}\} \dot\cup \\ & \{(P + u, u + P) \mid u \in \mathit{en}(Q) \wedge \ell(u) \notin \mathscr{B}{:}\} \dot\cup \\ & \{(t + u, u + t) \mid t \in \mathit{en}(P) \wedge u \in \mathit{en}(Q) \wedge \ell(t) = \ell(u) \in \mathscr{B}{:}\}\,. \end{aligned}$$

$R_{P,Q}$ relates transitions, i.e., derivations of transition triples, that are composed of the same sets of direct subderivations, even though their order is reversed.

Associativity of $+$, i.e., $(O + P) + Q \mathrel{\underline{\leftrightarrow}}_{ep} O + (P + Q)$: The relation

$$\{(I, I, \mathit{Id}_I) \mid I \in \mathbb{P}\} \dot\cup \{((O + P) + Q, O + (P + Q), R_{O,P,Q}) \mid O, P, Q \in \mathbb{P}\}$$

is an ep-bisimulation. Here Id_I and $R_{O,P,Q}$ are defined similarly to the previous case.

Commutativity of $|$, i.e., $P|Q \mathrel{\underline{\leftrightarrow}}_{ep} Q|P$: The relation $\{(P|Q, Q|P, R_{P,Q}) \mid P, Q \in \mathbb{P}\}$ is an ep-bisimulation. Here

$$\begin{aligned} R_{P,Q} =& \{(t|Q, Q|t) \mid t \in \mathit{en}(P) \wedge \ell(t) \notin \mathscr{B}! \dot\cup \mathscr{B}? \dot\cup \mathscr{B}{:}\} \dot\cup \\ & \{(P|u, u|P) \mid u \in \mathit{en}(Q) \wedge \ell(u) \notin \mathscr{B}! \dot\cup \mathscr{B}? \dot\cup \mathscr{B}{:}\} \dot\cup \\ & \{(t|u, u|t) \mid t \in \mathit{en}(P) \wedge u \in \mathit{en}(Q) \wedge \ell(t) = \overline{\ell(u)} \in \mathscr{C} \dot\cup \bar{\mathscr{C}} \dot\cup \mathscr{S} \dot\cup \bar{\mathscr{S}}\} \dot\cup \\ & \{(t|u, u|t) \mid t \in \mathit{en}(P) \wedge u \in \mathit{en}(Q) \wedge \\ & \qquad \exists\, b \in \mathscr{B}.\ \{\ell(t), \ell(u)\} \in \{\{b!, b?\}, \{b!, b{:}\}, \{b?\}, \{b?, b{:}\}, \{b{:}\}\}\}\,. \end{aligned}$$

Associativity of $|$, i.e., $(O|P)|Q \mathrel{\underline{\leftrightarrow}}_{ep} O|(P|Q)$: The relation

$$\{(O|P)|Q, O|(P|Q), R_{O,P,Q}) \mid O, P, Q \in \mathbb{P}\}$$

is an ep-bisimulation, where $R_{O,P,Q}$ is defined similarly to the previous case. ◀

Additionally, not only strong bisimilarity should be a congruence for all operators of ABCdE – which follows immediately from the De Simone format – but also our new ep-bisimilarity. This means that if two process terms are ep-bisimilar, then they are also ep-bisimilar in any context.

▶ **Theorem 22.** *Ep-bisimilarity is a congruence for all operators of ABCdE.*

We cannot get it directly from the existing meta-theory on structural operational semantics, as nobody has studied ep-bisimilarity before. As is standard, the proof is a case distinction on the type of the operator. For example, the case for action prefixing requires

$$P \underline{\leftrightarrow}_{ep} Q \Rightarrow \alpha.P \underline{\leftrightarrow}_{ep} \alpha.Q \text{ for } \alpha \in \mathit{Act}\,.$$

Such properties can be checked by inspecting the syntactic form of the transition rules, using structural induction. While proofs for some statements, such as the one for action prefixing, are merely a simple exercise, others require more care, including long and tedious case distinctions. A detailed proof of Theorem 22 can be found in Appendix A.

6 Failed Alternatives for Ep-Bisimulation

On an LTSS $(S, \mathit{Tr}, \mathit{source}, \mathit{target}, \ell, \rightsquigarrow)$ an ep-bisimulation has the type $S \times S \times \mathscr{P}(\mathit{Tr} \times \mathit{Tr})$. This is different from that of other classical bisimulations, which have the type $S \times S$. While developing ep-bisimulation we have also explored dozens of other candidates, many of them being of type $(S \times S) \uplus (\mathit{Tr} \times \mathit{Tr})$. The inclusion of a relation between transitions is necessary to reflect the concept of components or concurrency in one way or the other. One such candidate definition declares a relation $\mathscr{R} \subseteq (S \times S) \uplus (\mathit{Tr} \times \mathit{Tr})$ a valid bisimulation iff the set of triples

$$\{(p, q, R) \mid (p, q) \in \mathscr{R} \cap (S \times S) \wedge R = \mathscr{R} \cap (\mathit{en}(p) \times \mathit{en}(q))\}$$

is an ep-bisimulation. However, neither this candidate nor any of the others leads to a transitive notion of bisimilarity. The problem stems from systems, not hard to model in ABCdE, with multiple paths π_i from states p to p', such that a triple (p, q, R) in an ep-bisimulation $\mathscr{R}$ forces triples (p', q', R'_i) to be in $\mathscr{R}$ for multiple relations $R_i \subseteq \mathit{en}(p') \times \mathit{en}(q')$, depending on the chosen path π_i.

7 Related Work

Our LTSSs generalise the concurrent transition systems of [24]. There $t \rightsquigarrow_v u$ is written as $t{\uparrow}v = u$, and $\uparrow$ is a partial function rather than a relation, in the sense that for given t and v there can be at most one u with $t{\uparrow}v = u$. This condition is not satisfied by broadcast communication, which is one of the reasons we switched to the notation $t \rightsquigarrow_v u$. As an example, consider $b!|a.(b? + b?)$. The $b!$-transition after the a-transition has two variants, namely $\xrightarrow{b!}\mathbf{0}|(\xrightarrow{b?}\mathbf{0}{+}b?)$ and $\xrightarrow{b!}\mathbf{0}|(b?{+}\xrightarrow{b?}\mathbf{0})$. Another property of concurrent transition systems that is not maintained in our framework is the symmetry of the induced concurrency relation. Finally, [24] requires that $(v{\uparrow}t){\uparrow}(u{\uparrow}t) = (v{\uparrow}u){\uparrow}(t{\uparrow}u)$, the *cube axiom*, whereas we have so far not found reasons to restrict attention to processes satisfying this axiom. We are, however, open to the possibility that for future applications of LTSSs, some closure properties may be imposed on them.

In [1] a location-based bisimulation is proposed. It also keeps track of the components participating in transitions. The underlying model is quite different from ours, which makes it harder to formally argue that this notion of bisimilarity is incomparable to ours. We do not know yet whether it could be used to reason about justness.

8 Conclusion and Future Work

In related work, it has been argued that fairness assumptions used for verifying liveness properties of distributed systems are too strong or unrealistic [13, 4, 12]. As a consequence, justness, a minimal fairness assumption required for the verification of liveness properties, has been proposed. Unfortunately, all classical semantic equivalences, such as strong bisimilarity, fail to preserve justness.

In this paper, we have introduced labelled transition systems augmented by a successor relation, and, based on that, the concept of enabling preserving bisimilarity, a finer variant of strong bisimilarity. We have proven that this semantic equivalence is a congruence for all classical operators. As it also preserves justness, it is our belief that enabling preserving bisimilarity in combination with justness can and should be used for verifying liveness properties of large-scale distributed systems.

Casually speaking, ep-bisimilarity is strong bisimilarity augmented with the requirement that the relation between enabled transitions is inherited by successor transitions. A straightforward question is whether this feature can be combined with other semantic equivalences, such as weak bisimilarity or trace equivalence.

We have further shown how process algebras can be mapped into LTSSs. Of course, process algebra is only one of many formal frameworks for modelling concurrent systems. For accurately capturing causalities between event occurrences, models like Petri nets [22], event structures [25] or higher dimensional automata [21, 8] are frequently preferable. Part of future work is therefore to develop a formal semantics with respect to LTSSs for these frameworks.

In order to understand the scope of justness in real-world applications, we plan to study systems that depend heavily on liveness. As a starting point we plan to verify locks, such as ticket lock.

References

1 Gérard Boudol, Ilaria Castellani, Matthew Hennessy, and Astrid Kiehn. A theory of processes with localities. *Formal Aspects Comput.*, 6(2):165–200, 1994. doi:10.1007/BF01221098.

2 Sjoerd Cranen, Mohammad Reza Mousavi, and Michel A. Reniers. A rule format for associativity. In F. van Breugel and M. Chechik, editors, *Concurrency Theory (CONCUR '08)*, volume 5201 of LNCS, pages 447–461. Springer, 2008. doi:10.1007/978-3-540-85361-9_35.

3 Victor Dyseryn, Robert J. van Glabbeek, and Peter Höfner. Analysing mutual exclusion using process algebra with signals. In Kirstin Peters and Simone Tini, editors, Proc. Combined 24th International Workshop on *Expressiveness in Concurrency* and 14th Workshop on *Structural Operational Semantics*, volume 255 of *Electronic Proceedings in Theoretical Computer Science*, pages 18–34. Open Publishing Association, 2017. doi:10.4204/EPTCS.255.2.

4 Ansgar Fehnker, Robert J. van Glabbeek, Peter Höfner, Annabelle K. McIver, Marius Portmann, and Wee Lum Tan. A process algebra for wireless mesh networks used for modelling, verifying and analysing AODV. Technical Report 5513, NICTA, 2013. URL: http://arxiv.org/abs/1312.7645.

5 Nissim Francez. *Fairness*. Springer, 1986. doi:10.1007/978-1-4612-4886-6.

6 Robert J. van Glabbeek. Bisimulations for higher dimensional automata. Email message, July 7, 1991, 1991. URL: http://theory.stanford.edu/~rvg/hda.

7 Robert J. van Glabbeek. On specifying timeouts. In L. Aceto and A.D. Gordon, editors, Short Contributions from the Workshop on *Algebraic Process Calculi: The First Twenty Five Years and Beyond,* PA '05, Bertinoro, Italy, 2005, volume 162 of *Electronic Notes in Theoretical Computer Science*, pages 112–113. Elsevier, 2005. doi:10.1016/j.entcs.2005.12.083.

8 Robert J. van Glabbeek. On the expressiveness of higher dimensional automata. *Theoretical Computer Science*, 368(1-2):169–194, 2006. doi:10.1016/j.tcs.2006.06.024.

9 Robert J. van Glabbeek. Justness: A completeness criterion for capturing liveness properties. Technical report, Data61, CSIRO, 2019. Extended abstract in M. Bojańczyk & A. Simpson, editors: Proc. 22st International Conference on *Foundations of Software Science and Computation Structures* (FoSSaCS 2019); held as part of the European Joint Conferences on *Theory and Practice of Software* (ETAPS 2019), Prague, Czech Republic, 2019, LNCS 11425, Springer, pp. 505-522, doi:10.1007/978-3-030-17127-8_29. arXiv:1909.00286.

10 Robert J. van Glabbeek. Reactive temporal logic. In O. Dardha and J. Rot, editors, Proceedings Combined 27th International Workshop on *Expressiveness in Concurrency* and 17th Workshop on *Structural Operational Semantics,* Online, 31 August 2020, volume 322 of *Electronic Proceedings in Theoretical Computer Science*, pages 51–68. Open Publishing Association, 2020. doi:10.4204/EPTCS.322.6.

11 Robert J. van Glabbeek and Peter Höfner. CCS: it's not fair! *Acta Informatica*, 52(2-3):175–205, 2015. doi:10.1007/s00236-015-0221-6.

12 Robert J. van Glabbeek and Peter Höfner. Progress, fairness and justness in process algebra. Technical Report 8501, NICTA, 2015. arXiv:1501.03268.

13 Robert J. van Glabbeek and Peter Höfner. Progress, justness, and fairness. *ACM Computing Surveys*, 52(4), 2019. doi:10.1145/3329125.

14 Robert J. van Glabbeek, Peter Höfner, Marius Portmann, and Wee Lum Tan. Modelling and verifying the AODV routing protocol. *Distributed Computing*, 29(4):279–315, 2016. doi:10.1007/s00446-015-0262-7.

15 Robert J. van Glabbeek, Peter Höfner, and Weiyou Wang. Enabling preserving bisimulation equivalence, full version of the present paper, 2021. arXiv:2108.00142.

16 Eric Goubault and Thomas P. Jensen. Homology of higher dimensional automata. In W.R. Cleaveland, editor, *Proceedings CONCUR 92,* Stony Brook, NY, USA, volume 630 of LNCS, pages 254–268. Springer, 1992. doi:10.1007/BFb0084796.

17 Leslie Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, 1977. doi:10.1109/TSE.1977.229904.

18 Robin Milner. Operational and algebraic semantics of concurrent processes. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 19, pages 1201–1242. Elsevier Science Publishers B.V. (North-Holland), 1990. Alternatively see *Communication and Concurrency*, Prentice-Hall, Englewood Cliffs, 1989, of which an earlier version appeared as *A Calculus of Communicating Systems*, LNCS 92, Springer, 1980, doi:10.1007/3-540-10235-3.

19 Gordon D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60–61:17–139, 2004. Originally appeared in 1981. doi:10.1016/j.jlap.2004.05.001.

20 K. V. S. Prasad. A calculus of broadcasting systems. In Samson Abramsky and T. S. E. Maibaum, editors, TAPSOFT'91: Proceedings of the International Joint Conference on *Theory and Practice of Software Development,* Volume 1: *Colloquium on Trees in Algebra and Programming* (CAAP'91), volume 493 of LNCS, pages 338–358. Springer, 1991. doi:10.1007/3-540-53982-4_19.

21 Vaughan R. Pratt. Modeling concurrency with geometry. In *Principles of Programming Languages (PoPL'91)*, pages 311–322, 1991. doi:10.1145/99583.99625.

22 Wolfgang Reisig. *Understanding Petri Nets — Modeling Techniques, Analysis Methods, Case Studies.* Springer, 2013. doi:10.1007/978-3-642-33278-4.

23 Robert de Simone. Higher-level synchronising devices in MEIJE-SCCS. *Theoretical Comp. Science*, 37:245–267, 1985. doi:10.1016/0304-3975(85)90093-3.

24 Eugine W. Stark. Concurrent transition systems. *Theoretical Computer Science*, 64:221–269, 1989. doi:10.1016/0304-3975(89)90050-9.

25 Glynn Winskel. Event structures. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II*, volume 255 of LNCS, pages 325–392. Springer, 1987. doi:10.1007/3-540-17906-2_31.

A Congruence Proofs

Ep-bisimilarity is a congruence for all operators of ABCdE iff Propositions 23–28 below hold. We prove them one by one.

▶ **Proposition 23.** If $P \leftrightarroweq_{ep} Q$ and $\alpha \in Act$ then $\alpha.P \leftrightarroweq_{ep} \alpha.Q$.

Proof. A transition enabled in $\alpha.P$ is either $\xrightarrow{\alpha}P$ or $b{:}\alpha.P$ for some $b \in \mathscr{B}$ with $\alpha \neq b?$.

Let $\mathscr{R} \subseteq \mathbb{P} \times \mathbb{P} \times \mathscr{P}(Tr \times Tr)$ be the smallest relation satisfying

1. if $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$ then $(P, Q, R) \in \mathscr{R}$,
2. if $(P, Q, R) \in \mathscr{R}$ and $\alpha \in Act$ then $(\alpha.P, \alpha.Q, \alpha.R) \in \mathscr{R}$, where

$$\alpha.R := \{(\xrightarrow{\alpha}P, \xrightarrow{\alpha}Q)\} \cup \{(b{:}\alpha.P, b{:}\alpha.Q) \mid b \in \mathscr{B} \wedge \alpha \neq b?\}\,.$$

It suffices to show that $\mathscr{R}$ is an ep-bisimulation. I.e., all entries in $\mathscr{R}$ satisfy the requirements of Definition 8. We proceed by structural induction.

Induction base: Suppose $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$. Since $\mathscr{R}' \subseteq \mathscr{R}$, all requirements of Definition 8 are satisfied.

Induction step: Suppose $(P, Q, R) \in \mathscr{R}$ satisfies all requirements of Definition 8, we prove that (P', Q', R'), where $P' = \alpha.P$, $Q' = \alpha.Q$, and $R' = \alpha.R$, also satisfies those requirements. This follows directly with Definitions 8 and 20. ◀

▶ **Proposition 24.** If $P_\text{L} \leftrightarroweq_{ep} Q_\text{L}$ and $P_\text{R} \leftrightarroweq_{ep} Q_\text{R}$ then $P_\text{L} + P_\text{R} \leftrightarroweq_{ep} Q_\text{L} + Q_\text{R}$.

Proof. A transition enabled in $P + Q$ is either

- $t + Q$ for some $t \in en(P)$ with $\ell(t) \notin \mathscr{B}{:}$,
- $P + u$ for some $u \in en(Q)$ with $\ell(u) \notin \mathscr{B}{:}$, or
- $t + u$ for some $t \in en(P)$ and $u \in en(Q)$ with $\ell(t) = \ell(u) \in \mathscr{B}{:}$.

Let $\mathscr{R} \subseteq \mathbb{P} \times \mathbb{P} \times \mathscr{P}(Tr \times Tr)$ be the smallest relation satisfying

1. if $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$ then $(P, Q, R) \in \mathscr{R}$,
2. if $(P_\text{L}, Q_\text{L}, R_\text{L}), (P_\text{R}, Q_\text{R}, R_\text{R}) \in \mathscr{R}$ then $(P_\text{L} + P_\text{R}, Q_\text{L} + Q_\text{R}, R_\text{L} + R_\text{R}) \in \mathscr{R}$, where

$$\begin{aligned} R_\text{L} + R_\text{R} :=& \{(t + P_\text{R}, v + Q_\text{R}) \mid t\ R_\text{L}\ v \wedge \ell(t) \notin \mathscr{B}{:}\} \cup \\ & \{(P_\text{L} + u, Q_\text{L} + w) \mid u\ R_\text{R}\ w \wedge \ell(u) \notin \mathscr{B}{:}\} \cup \\ & \{(t + u, v + w) \mid t\ R_\text{L}\ v \wedge u\ R_\text{R}\ w \wedge \ell(t) = \ell(u) \in \mathscr{B}{:}\}\,. \end{aligned}$$

It suffices to show that $\mathscr{R}$ is an ep-bisimulation. I.e., all entries in $\mathscr{R}$ satisfy the requirements of Definition 8. We proceed by structural induction.

Induction base: Suppose $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$. Since $\mathscr{R}' \subseteq \mathscr{R}$, all requirements of Definition 8 are satisfied.

Induction step: Suppose $(P_\text{L}, Q_\text{L}, R_\text{L}), (P_\text{R}, Q_\text{R}, R_\text{R}) \in \mathscr{R}$ satisfy all requirements of Definition 8, we prove that (P, Q, R), where $P = P_\text{L} + P_\text{R}$, $Q = Q_\text{L} + Q_\text{R}$ and $R = R_\text{L} + R_\text{R}$, also satisfies those requirements.

$R \subseteq en(P) \times en(Q)$ follows from $R_\text{L} \subseteq en(P_\text{L}) \times en(Q_\text{L})$ and $R_\text{R} \subseteq en(P_\text{R}) \times en(Q_\text{R})$.

Requirement 1.a: It suffices to find, for each $\chi \in en(P)$, a $\zeta \in en(Q)$ with $\chi\ R\ \zeta$.

1. Suppose $\chi = t + P_\text{R}$ for some $t \in en(P_\text{L})$ with $\ell(t) \notin \mathscr{B}{:}$.
   We obtain $v \in en(Q_\text{L})$ with $t\ R_\text{L}\ v$, and pick $\zeta = v + Q_\text{R}$.
2. Suppose $\chi = P_\text{L} + u$ for some $u \in en(P_\text{R})$ with $\ell(u) \notin \mathscr{B}{:}$.
   We obtain $w \in en(Q_\text{R})$ with $u\ R_\text{R}\ w$, and pick $\zeta = Q_\text{L} + w$.

3. Suppose $\chi = t + u$ for some $t \in en(P_{\text{L}})$ and $u \in en(P_{\text{R}})$ with $\ell(t) = \ell(u) \in \mathscr{B}{:}$.
   We obtain $v \in en(Q_{\text{L}})$ and $w \in en(Q_{\text{R}})$ with $t\ R_{\text{L}}\ v$ and $u\ R_{\text{R}}\ w$, and pick $\zeta = v + w$.

In all cases, $\zeta \in en(Q)$ and $\chi\ R\ \zeta$ hold trivially.

Requirement 1.b: The proof is similar to that of Requirement 1.(a) and is omitted.

Requirement 1.c: This follows directly from the observations that

- $\ell(t) = \ell(v) \implies \ell(t + P_{\text{R}}) = \ell(v + Q_{\text{R}})$,
- $\ell(u) = \ell(w) \implies \ell(P_{\text{L}} + u) = \ell(Q_{\text{L}} + w)$, and
- $\ell(t) = \ell(v) \wedge \ell(u) = \ell(w) \implies \ell(t + u) = \ell(v + w)$;

provided that the composed transitions exist.

Requirement 2: It suffices to find, for arbitrary Υ, Υ' with $\Upsilon\ R\ \Upsilon'$, an R' with $(target(\Upsilon), target(\Upsilon'), R') \in \mathscr{R}$, such that

(a) for arbitrary χ, χ' with $\chi\ R\ \chi'$ and $\chi \leadsto_{\Upsilon} \zeta$, we can find a ζ' with $\chi' \leadsto_{\Upsilon'} \zeta'$ and $\zeta\ R'\ \zeta'$,

(b) for arbitrary χ, χ' with $\chi\ R\ \chi'$ and $\chi' \leadsto_{\Upsilon'} \zeta'$, we can find a ζ with $\chi \leadsto_{\Upsilon} \zeta$ and $\zeta\ R'\ \zeta'$.

Below we focus merely on (a), as (b) will follow by symmetry.

Suppose $\ell(\Upsilon) \in \mathscr{B}{:} \cup \bar{\mathscr{S}}$. Pick $R' = R$. Then $(target(\Upsilon), target(\Upsilon'), R') = (P, Q, R) \in \mathscr{R}$. From $\chi \leadsto_{\Upsilon} \zeta$ we have $\zeta = \chi$. Pick $\zeta' = \chi'$. Then $\chi' \leadsto_{\Upsilon'} \zeta'$. $\zeta\ R'\ \zeta'$ is given by $\chi\ R\ \chi'$.

We further split the cases when $\ell(\Upsilon) \in Act$.

1. Suppose $\Upsilon = v + P_{\text{R}}$ and $\Upsilon' = v' + Q_{\text{R}}$ with $v\ R_{\text{L}}\ v'$. We obtain R'_{L} that satisfies Requirement 2 with respect to v and v'. Pick $R' = R'_{\text{L}}$. Then $(target(\Upsilon), target(\Upsilon'), R') = (target(v), target(v'), R'_{\text{L}}) \in \mathscr{R}$.
   a. Suppose $\chi = t + P_{\text{R}}$ and $\chi' = t' + Q_{\text{R}}$ with $t\ R_{\text{L}}\ t'$. From $\chi \leadsto_{\Upsilon} \zeta$ we have $t \leadsto_{v} \zeta$. Then we obtain x' with $t' \leadsto_{v'} x'$ and $\zeta\ R'_{\text{L}}\ x'$. Pick $\zeta' = x'$. Then $\chi' \leadsto_{\Upsilon'} \zeta'$ follows from $t' \leadsto_{v'} x'$.
   b. Suppose $\chi = P_{\text{L}} + u$ and $\chi' = Q_{\text{L}} + u'$ with $u\ R_{\text{R}}\ u'$. We obtain x' with $\zeta\ R'_{\text{L}}\ x'$ and pick $\zeta' = x'$. From $\chi \leadsto_{\Upsilon} \zeta$ we have $\ell(\chi) = b?$ and $\ell(\zeta) \in \{b?, b{:}\}$ for some $b \in \mathscr{B}$. Then $\chi' \leadsto_{\Upsilon'} \zeta'$ follows from $\ell(\chi') = b?$ and $\ell(x') \in \{b?, b{:}\}$.
   c. Suppose $\chi = t + u$ and $\chi' = t' + u'$ with $t\ R_{\text{L}}\ t'$ and $u\ R_{\text{R}}\ u'$. From $\chi \leadsto_{\Upsilon} \zeta$ we have $t \leadsto_{v} \zeta$. Then we obtain x' with $t' \leadsto_{v'} x'$ and $\zeta\ R'_{\text{L}}\ x'$. Pick $\zeta' = x'$. Then $\chi' \leadsto_{\Upsilon'} \zeta'$ follows from $t' \leadsto_{v'} x'$.

   In all cases, $\zeta\ R'\ \zeta'$ is given by $\zeta\ R'_{\text{L}}\ x'$.
2. Suppose $\Upsilon = P_{\text{L}} + w$ and $\Upsilon' = Q_{\text{L}} + w'$ with $w\ R_{\text{R}}\ w'$. The proof is similar to that of the previous case. ◀

▶ **Proposition 25.** If $P_{\text{L}} \leftrightarroweq_{ep} Q_{\text{L}}$ and $P_{\text{R}} \leftrightarroweq_{ep} Q_{\text{R}}$ then $P_{\text{L}}|P_{\text{R}} \leftrightarroweq_{ep} Q_{\text{L}}|Q_{\text{R}}$.

Proof. A transition enabled in $P|Q$ is either

- $t|Q$ for some $t \in en(P)$ with $\ell(t) \notin \mathscr{B}! \cup \mathscr{B}? \cup \mathscr{B}{:}$,
- $P|u$ for some $u \in en(Q)$ with $\ell(u) \notin \mathscr{B}! \cup \mathscr{B}? \cup \mathscr{B}{:}$,
- $t|u$ for some $t \in en(P)$ and $u \in en(Q)$ with $\ell(t) = \overline{\ell(u)} \in \mathscr{C} \cup \bar{\mathscr{C}} \cup \mathscr{S} \cup \bar{\mathscr{S}}$, or
- $t|u$ for some $t \in en(P)$ and $u \in en(Q)$ with $\{\ell(t), \ell(u)\} \in \{\{b!, b?\}, \{b!, b{:}\}, \{b?\}, \{b?, b{:}\}, \{b{:}\}\}$ for some $b \in \mathscr{B}$.

Let $\mathscr{R} \subseteq \mathbb{P} \times \mathbb{P} \times \mathscr{P}(Tr \times Tr)$ be the smallest relation satisfying

1. if $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$ then $(P, Q, R) \in \mathscr{R}$,

2. if $(P_{\mathrm{L}}, Q_{\mathrm{L}}, R_{\mathrm{L}}), (P_{\mathrm{R}}, Q_{\mathrm{R}}, R_{\mathrm{R}}) \in \mathscr{R}$ then $(P_{\mathrm{L}}|P_{\mathrm{R}}, Q_{\mathrm{L}}|Q_{\mathrm{R}}, R_{\mathrm{L}}|R_{\mathrm{R}}) \in \mathscr{R}$, where

$$\begin{aligned} R_{\mathrm{L}}|R_{\mathrm{R}} :=& \{(t|P_{\mathrm{R}}, v|Q_{\mathrm{R}}) \mid t\ R_{\mathrm{L}}\ v \wedge \ell(t) \notin \mathscr{B}! \mathbin{\dot\cup} \mathscr{B}? \mathbin{\dot\cup} \mathscr{B}{:}\} \mathbin{\dot\cup} \\ & \{(P_{\mathrm{L}}|u, Q_{\mathrm{L}}|w) \mid u\ R_{\mathrm{R}}\ w \wedge \ell(u) \notin \mathscr{B}! \mathbin{\dot\cup} \mathscr{B}? \mathbin{\dot\cup} \mathscr{B}{:}\} \mathbin{\dot\cup} \\ & \{(t|u, v|w) \mid t\ R_{\mathrm{L}}\ v \wedge u\ R_{\mathrm{R}}\ w \wedge \ell(t) = \overline{\ell(u)} \in \mathscr{C} \mathbin{\dot\cup} \bar{\mathscr{C}} \mathbin{\dot\cup} \mathscr{S} \mathbin{\dot\cup} \bar{\mathscr{S}}\} \mathbin{\dot\cup} \\ & \{(t|u, v|w) \mid t\ R_{\mathrm{L}}\ v \wedge u\ R_{\mathrm{R}}\ w \wedge \\ & \qquad\qquad \exists\, b \in \mathscr{B}.\ \{\ell(t), \ell(u)\} \in \{\{b!, b?\}, \{b!, b{:}\}, \{b?\}, \{b?, b{:}\}, \{b{:}\}\}\}\,. \end{aligned}$$

It suffices to show that $\mathscr{R}$ is an ep-bisimulation. I.e., all entries in $\mathscr{R}$ satisfy the requirements of Definition 8. We proceed by structural induction.

Induction base: Suppose $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$. Since $\mathscr{R}' \subseteq \mathscr{R}$, all requirements of Definition 8 are satisfied.

Induction step: Suppose $(P_{\mathrm{L}}, Q_{\mathrm{L}}, R_{\mathrm{L}}), (P_{\mathrm{R}}, Q_{\mathrm{R}}, R_{\mathrm{R}}) \in \mathscr{R}$ satisfy all requirements of Definition 8, we prove that (P, Q, R), where $P = P_{\mathrm{L}}|P_{\mathrm{R}}$, $Q = Q_{\mathrm{L}}|Q_{\mathrm{R}}$ and $R = R_{\mathrm{L}}|R_{\mathrm{R}}$, also satisfies those requirements.

$R \subseteq \mathit{en}(P) \times \mathit{en}(Q)$ follows from $R_{\mathrm{L}} \subseteq \mathit{en}(P_{\mathrm{L}}) \times \mathit{en}(Q_{\mathrm{L}})$ and $R_{\mathrm{R}} \subseteq \mathit{en}(P_{\mathrm{R}}) \times \mathit{en}(Q_{\mathrm{R}})$.

Requirement 1.a: It suffices to find, for each $\chi \in \mathit{en}(P)$, a $\zeta \in \mathit{en}(Q)$ with $\chi\ R\ \zeta$.

1. Suppose $\chi = t|P_{\mathrm{R}}$ for some $t \in \mathit{en}(P_{\mathrm{L}})$ with $\ell(t) \notin \mathscr{B}! \mathbin{\dot\cup} \mathscr{B}? \mathbin{\dot\cup} \mathscr{B}{:}$.
   We obtain $v \in \mathit{en}(Q_{\mathrm{L}})$ with $t\ R_{\mathrm{L}}\ v$ and pick $\zeta = v|Q_{\mathrm{R}}$.
2. Suppose $\chi = P_{\mathrm{L}}|u$ for some $u \in \mathit{en}(P_{\mathrm{R}})$ with $\ell(u) \notin \mathscr{B}! \mathbin{\dot\cup} \mathscr{B}? \mathbin{\dot\cup} \mathscr{B}{:}$.
   We obtain $w \in \mathit{en}(Q_{\mathrm{R}})$ with $u\ R_{\mathrm{R}}\ w$ and pick $\zeta = Q_{\mathrm{L}}|w$.
3. Suppose $\chi = t|u$ for some $t \in \mathit{en}(P_{\mathrm{L}})$, $u \in \mathit{en}(P_{\mathrm{R}})$ with $\ell(t) = \overline{\ell(u)} \in \mathscr{C} \mathbin{\dot\cup} \bar{\mathscr{C}} \mathbin{\dot\cup} \mathscr{S} \mathbin{\dot\cup} \bar{\mathscr{S}}$.
   We obtain $v \in \mathit{en}(Q_{\mathrm{L}})$ and $w \in \mathit{en}(Q_{\mathrm{R}})$ with $t\ R_{\mathrm{L}}\ v$ and $u\ R_{\mathrm{R}}\ w$, and pick $\zeta = v|w$.
4. Suppose $\chi = t|u$ for some $t \in \mathit{en}(P_{\mathrm{L}})$ and $u \in \mathit{en}(P_{\mathrm{R}})$ with
   $\{\ell(t), \ell(u)\} \in \{\{b!, b?\}, \{b!, b{:}\}, \{b?\}, \{b?, b{:}\}, \{b{:}\}\}$ for some $b \in \mathscr{B}$.
   We obtain $v \in \mathit{en}(Q_{\mathrm{L}})$ and $w \in \mathit{en}(Q_{\mathrm{R}})$ with $t\ R_{\mathrm{L}}\ v$ and $u\ R_{\mathrm{R}}\ w$, and pick $\zeta = v|w$.

In all cases, $\zeta \in \mathit{en}(Q)$ and $\chi\ R\ \zeta$ hold trivially.

Requirement 1.b: The proof is similar to that of Requirement 1.(a) and is omitted.

Requirement 1.c: This follows directly from the observation that

- $\ell(t) = \ell(v) \implies \ell(t|P_{\mathrm{R}}) = \ell(v|Q_{\mathrm{R}})$,
- $\ell(u) = \ell(w) \implies \ell(P_{\mathrm{L}}|u) = \ell(Q_{\mathrm{L}}|w)$, and
- $\ell(t) = \ell(v) \wedge \ell(u) = \ell(w) \implies \ell(t|u) = \ell(v|w)$;

provided that the composed transitions exist.

Requirement 2: It suffices to find, for arbitrary Υ, Υ' with $\Upsilon\ R\ \Upsilon'$, an R' with $(\mathit{target}(\Upsilon), \mathit{target}(\Upsilon'), R') \in \mathscr{R}$, such that

(a) for arbitrary χ, χ' with $\chi\ R\ \chi'$ and $\chi \rightsquigarrow_{\Upsilon} \zeta$, we can find a ζ' with $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ and $\zeta\ R'\ \zeta'$,

(b) for arbitrary χ, χ' with $\chi\ R\ \chi'$ and $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$, we can find a ζ with $\chi \rightsquigarrow_{\Upsilon} \zeta$ and $\zeta\ R'\ \zeta'$.

Below we focus merely on (a), as (b) will follow by symmetry.

1. Suppose $\Upsilon = v|P_{\mathrm{R}}$ and $\Upsilon' = v'|Q_{\mathrm{R}}$ with $v\ R_{\mathrm{L}}\ v'$. We obtain R'_{L} that satisfies Requirement 2 with respect to v and v'. Pick $R' = R'_{\mathrm{L}}|R_{\mathrm{R}}$. Then $(\mathit{target}(\Upsilon), \mathit{target}(\Upsilon'), R') = (\mathit{target}(v)|P_{\mathrm{R}}, \mathit{target}(v')|Q_{\mathrm{R}}, R'_{\mathrm{L}}|R_{\mathrm{R}}) \in \mathscr{R}$.
   a. Suppose $\chi = t|P_{\mathrm{R}}$ and $\chi' = t'|Q_{\mathrm{R}}$ with $t\ R_{\mathrm{L}}\ t'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $\zeta = x|P_{\mathrm{R}}$ for some x with $t \rightsquigarrow_{v} x$. Then we obtain x' with $t' \rightsquigarrow_{v'} x'$ and $x\ R'_{\mathrm{L}}\ x'$. Pick $\zeta' = x'|Q_{\mathrm{R}}$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows from $t' \rightsquigarrow_{v'} x'$; $\zeta\ R'\ \zeta'$ is given by $x\ R'_{\mathrm{L}}\ x'$.
   b. Suppose $\chi = P_{\mathrm{L}}|u$ and $\chi' = Q_{\mathrm{L}}|u'$ with $u\ R_{\mathrm{R}}\ u'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $\zeta = \mathit{target}(v)|u$. Pick $\zeta' = \mathit{target}(v')|u'$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows directly; $\zeta\ R'\ \zeta'$ is given by $u\ R_{\mathrm{R}}\ u'$.

   c. Suppose $\chi = t|u$ and $\chi' = t'|u'$ with $t \; R_{\text{L}} \; t'$ and $u \; R_{\text{R}} \; u'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $\zeta = x|u$ for some x with $t \rightsquigarrow_v x$. Then we obtain x' with $t' \rightsquigarrow_{v'} x'$ and $x \; R'_{\text{L}} \; x'$. Pick $\zeta' = x'|u'$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows from $t' \rightsquigarrow_{v'} x'$; $\zeta \; R' \; \zeta'$ is given by $x \; R'_{\text{L}} \; x'$ and $u \; R_{\text{R}} \; u'$.

2. Suppose $\Upsilon = P_{\text{L}}|w$ and $\Upsilon' = Q_{\text{L}}|w'$ with $w \; R_{\text{R}} \; w'$. The proof is similar to that of the previous case.
3. Suppose $\Upsilon = v|w$ and $\Upsilon' = v'|w'$ with $v \; R_{\text{L}} \; v'$ and $w \; R_{\text{R}} \; w'$. We obtain R'_{L} that satisfies Requirement 2 with respect to v and v', and R'_{R} that satisfies Requirement 2 with respect to w and w'. Pick $R' = R'_{\text{L}}|R'_{\text{R}}$. Then $(\mathit{target}(\Upsilon), \mathit{target}(\Upsilon'), R') = (\mathit{target}(v)|\mathit{target}(w), \mathit{target}(v')|\mathit{target}(w'), R'_{\text{L}}|R'_{\text{R}}) \in \mathscr{R}$.
   a. Suppose $\chi = t|P_{\text{R}}$ and $\chi' = t'|Q_{\text{R}}$ with $t \; R_{\text{L}} \; t'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $\zeta = x|\mathit{target}(w)$ for some x with $t \rightsquigarrow_v x$. Then we obtain x' with $t' \rightsquigarrow_{v'} x'$ and $x \; R'_{\text{L}} \; x'$. Pick $\zeta' = x'|\mathit{target}(w')$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows from $t' \rightsquigarrow_{v'} x'$; $\zeta \; R' \; \zeta'$ is given by $x \; R'_{\text{L}} \; x'$.
   b. Suppose $\chi = P_{\text{L}}|u$ and $\chi' = Q_{\text{L}}|u'$ with $u \; R_{\text{R}} \; u'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $\zeta = \mathit{target}(v)|y$ for some y with $u \rightsquigarrow_w y$. Then we obtain y' with $u' \rightsquigarrow_{w'} y'$ and $y \; R'_{\text{R}} \; y'$. Pick $\zeta' = \mathit{target}(v')|y'$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows from $u' \rightsquigarrow_{w'} y'$; $\zeta \; R' \; \zeta'$ is given by $y \; R'_{\text{R}} \; y'$.
   c. Suppose $\chi = t|u$ and $\chi' = t'|u'$ with $t \; R_{\text{L}} \; t'$ and $u \; R_{\text{R}} \; u'$. From $\chi \rightsquigarrow_{\Upsilon} \zeta$ we have $\zeta = x|y$ for some x, y with $t \rightsquigarrow_v x$ and $u \rightsquigarrow_w y$. Then we obtain x', y' with $t' \rightsquigarrow_{v'} x'$, $u' \rightsquigarrow_{w'} y'$, $x \; R'_{\text{L}} \; x'$, and $y \; R'_{\text{R}} \; y'$. Pick $\zeta' = x'|y'$. Then $\chi' \rightsquigarrow_{\Upsilon'} \zeta'$ follows from $t' \rightsquigarrow_{v'} x'$ and $u' \rightsquigarrow_{w'} y'$; $\zeta \; R' \; \zeta'$ is given by $x \; R'_{\text{L}} \; x'$ and $y \; R'_{\text{R}} \; y'$. ◀

▶ **Proposition 26.** If $P \leftrightarroweq_{ep} Q$ and $L \subseteq \mathscr{C} \uplus \mathscr{S}$ then $P\backslash L \leftrightarroweq_{ep} Q\backslash L$.

Proof. A transition enabled in $P\backslash L$ is $t\backslash L$ for some $t \in \mathit{en}(P)$ with $\ell(t) \notin L \uplus \overline{L}$.

Let $\mathscr{R} \subseteq \mathbb{P} \times \mathbb{P} \times \mathscr{P}(\mathit{Tr} \times \mathit{Tr})$ be the smallest relation satisfying

1. if $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$ then $(P, Q, R) \in \mathscr{R}$,
2. if $(P, Q, R) \in \mathscr{R}$ and $L \subseteq \mathscr{C} \uplus \mathscr{S}$ then $(P\backslash L, Q\backslash L, R\backslash L) \in \mathscr{R}$, where

$$R\backslash L := \{(t\backslash L, v\backslash L) \mid t \; R \; v \wedge \ell(t) \notin L \uplus \overline{L}\} .$$

It suffices to show that $\mathscr{R}$ is an ep-bisimulation. I.e., all entries in $\mathscr{R}$ satisfy the requirements of Definition 8. We proceed by structural induction.

Induction base: Suppose $(P, Q, R) \in \mathscr{R}'$ for some ep-bisimulation $\mathscr{R}'$. Since $\mathscr{R}' \subseteq \mathscr{R}$, all requirements of Definition 8 are satisfied.

Induction step: Suppose $(P, Q, R) \in \mathscr{R}$ satisfies all requirements of Definition 8, we prove that (P', Q', R'), where $P' = P\backslash L$, $Q' = Q\backslash L$, and $R' = R\backslash L$, also satisfies those requirements. This follows directly with Definitions 8 and 20. ◀

▶ **Proposition 27.** If $P \leftrightarroweq_{ep} Q$ and f is a relabelling then $P[f] \leftrightarroweq_{ep} Q[f]$.

Proof. An easy structural induction on the structure of P; similar to the proof for restriction (Proposition 26). ◀

▶ **Proposition 28.** If $P \leftrightarroweq_{ep} Q$ and $s \in \mathscr{S}$ $P\hat{\ }s \leftrightarroweq_{ep} Q\hat{\ }s$.

Proof. An easy structural induction on the structure of P; similar to the proof for restriction (Proposition 26). ◀

Sized Types with Usages for Parallel Complexity of Pi-Calculus Processes

Patrick Baillot
Univ Lyon, CNRS, ENS de Lyon, Universite Claude-Bernard Lyon 1, LIP, F-69342, France

Alexis Ghyselen
Univ Lyon, CNRS, ENS de Lyon, Universite Claude-Bernard Lyon 1, LIP, F-69342, France

Naoki Kobayashi
The University of Tokyo, Japan

Abstract

We address the problem of analysing the complexity of concurrent programs written in Pi-calculus. We are interested in parallel complexity, or span, understood as the execution time in a model with maximal parallelism. A type system for parallel complexity has been recently proposed by the first two authors but it is too imprecise for non-linear channels and cannot analyse some concurrent processes. Aiming for a more precise analysis, we design a type system which builds on the concepts of sized types and usages. The sized types allow us to parametrize the complexity by the size of inputs, and the usages allow us to achieve a kind of rely-guarantee reasoning on the timing each process communicates with its environment. We prove that our new type system soundly estimates the parallel complexity, and show through examples that it is often more precise than the previous type system of the first two authors.

2012 ACM Subject Classification Theory of computation → Type structures; Theory of computation → Process calculi; Software and its engineering → Software verification

Keywords and phrases Type Systems, Pi-calculus, Process Calculi, Complexity Analysis, Usages, Sized Types

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.34

Related Version *Extended Version*: `https://arxiv.org/abs/2104.07293`

Funding This work was partially supported by the LABEX MILYON (ANR-10-LABX-0070) of Universite de Lyon and JSPS KAKENHI Grant Number JP20H05703.

Acknowledgements We would like to thank anonymous referees for useful comments.

1 Introduction

Static analysis of complexity is a classic topic of program analysis, and various approaches to the complexity analysis, including type-based ones [3, 6, 17–19, 21], have been studied so far. The complexity analysis of concurrent programs has been, however, much less studied.

In this paper, we are interested in analysing the parallel complexity (also called *span*) of the π-calculus, i.e., the maximal parallelized execution time under the assumption that an unlimited number of processors are available [4]. The parallel complexity should be parametrized by the size of inputs. Following the success of previous studies on the complexity analysis of sequential programs [3, 6, 17–19, 21] and those on the analysis of other properties on concurrent programs (e.g. [11, 13]), deadlock-freedom and livelock-freedom (e.g. [22, 27, 30]; see [24] for a survey), we take a type-based approach.

The first two authors [4] have actually proposed a type-based analysis of the parallel complexity already. However, as stressed by the authors, even though their type system for span is useful for analysing the complexity of some parallel programs, it fails to type-

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 34; pp. 34:1–34:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

check some examples of common concurrent programs, like semaphores. It is based on a combination of sized types and input/output types, in order to account suitably for the behaviour of channels w.r.t. reception and emission.

In the present paper we design a type system for span which can deal with a much wider range of concurrent computation patterns including the semaphores. For that, we take inspiration from the notion of *type usage*, which has been introduced and explored in [27, 30], initially to guarantee absence of deadlock during execution. Type usages are a generalization of input/output types, and describe how each channel is used for input and output. Unlike the original notion of usages [27, 30], our usages are annotated with time intervals, which are used for a kind of rely-guarantee reasoning, like “assuming that a message from the environment arrives during the time interval $[I_1, J_1]$, the process sends back a message during the interval $[I_2, J_2]$”; such reasoning is crucial for analyzing the parallel complexity precisely and in a compositional manner. We formalize the type system with usages and prove that it soundly estimates the parallel complexity.

Contributions. The contributions of this paper are as follows. (i) The formalization of the new type system for parallel complexity built on the new notion of usages: our usages are quite different from the original ones, and properly defining them (including operations on time intervals, usage reductions, and the notion of reliable usages) is non-trivial. (ii) The proofs of type preservation and complexity soundness: thanks to the careful definition of new usages, the proofs are actually quite natural, despite the expressiveness of the type system. (iii) Examples to demonstrate the precision and expressive power of our new type system.

Paper outline. We introduce in Sect. 2 the π-calculus and the notion of parallel complexity we consider. Sect. 3 is devoted to the definition of types with usages. Then in Sect. 4 we prove the main result of this paper, the complexity soundness, and provide some examples. Finally, related work is discussed in Sect. 5.

2 The Pi-calculus with Semantics for Span

In this section, we review the definitions of the π-calculus and its parallel complexity [4].

2.1 Syntax and Standard Semantics for Pi-Calculus

We consider a synchronous π-calculus, with a constructor `tick` that generates the time complexity. The sets of *variables*, *expressions* and *processes* are defined by:

$$\begin{aligned} v \text{ (variables)} &:= x, y, z \mid a, b, c \qquad e \text{ (expressions)} := v \mid 0 \mid \mathtt{s}(e) \\ P \text{ (processes)} &:= 0 \mid (P \mid Q) \mid a(\widetilde{v}).P \mid !a(\widetilde{v}).P \mid \overline{a}\langle\widetilde{e}\rangle.P \mid (\nu a)P \\ &\mid \mathtt{match}\ e\ \{\mathtt{case}\ 0 \mapsto P; \mathtt{case}\ \mathtt{s}(x) \mapsto Q\} \mid \mathtt{tick}.P \end{aligned}$$

We use x, y, z as meta-variables for integer variables, and a, b, c as those for channel names. The notation $\widetilde{v}$ stands for a sequence of variables $v_1, v_2, \ldots, v_k$. Similarly, $\widetilde{e}$ denotes a sequence of expressions. We work up to α-renaming, and write $P[\widetilde{v} := \widetilde{e}]$ to denote the substitution of $\widetilde{e}$ for the free variables $\widetilde{v}$ in P. For simplicity, we only consider integers as base types below, but the results can be generalized to other algebraic data-types such as lists or booleans.

Intuitively, $P \mid Q$ stands for the parallel composition of P and Q. The process $a(\widetilde{v}).P$ represents an input: it stands for the reception on the channel a of a tuple of values identified by the variables $\widetilde{v}$ in the continuation P. The process $!a(\widetilde{v}).P$ is a replicated version of $a(\widetilde{v}).P$:

$$\frac{}{(n : a(\widetilde{v}).P) \mid (m : \overline{a}\langle\widetilde{e}\rangle.Q) \Rightarrow \max(m,n) : (P[\widetilde{v} := \widetilde{e}] \mid Q)} \qquad \frac{}{\texttt{tick}.P \Rightarrow 1 : P}$$

$$\frac{}{(n : !a(\widetilde{v}).P) \mid (m : \overline{a}\langle\widetilde{e}\rangle.Q) \Rightarrow (n : !a(\widetilde{v}).P) \mid (\max(m,n) : (P[\widetilde{v} := \widetilde{e}] \mid Q))}$$

$$\frac{}{\texttt{match } 0 \ \{\texttt{case } 0 \mapsto P; \texttt{case s}(x) \mapsto Q\} \Rightarrow P}$$

$$\frac{}{\texttt{match s}(e) \ \{\texttt{case } 0 \mapsto P; \texttt{case s}(x) \mapsto Q\} \Rightarrow Q[x := e]}$$

$$\frac{P \Rightarrow Q}{P \mid R \Rightarrow Q \mid R} \qquad \frac{P \Rightarrow Q}{(\nu a)P \Rightarrow (\nu a)Q} \qquad \frac{P \Rightarrow Q}{(n : P) \Rightarrow (n : Q)}$$

$$\frac{P \equiv P' \qquad P' \Rightarrow Q' \qquad Q' \equiv Q}{P \Rightarrow Q}$$

Figure 1 Reduction Rules for Annotated Processes.

it behaves like an infinite number of $a(\widetilde{v}).P$ in parallel. The process $\overline{a}\langle\widetilde{e}\rangle.P$ represents an output: it sends a sequence of expressions $\widetilde{e}$ on the channel a, and continues as P. We often omit 0 and just write $\overline{a}\langle\rangle$ for $\overline{a}\langle\rangle.0$. A process $(\nu a)P$ dynamically creates a new channel name a and then proceeds as P. We also have standard pattern matching on data types, and finally, the `tick` constructor incurs a cost of one in complexity but has no semantic relevance. This constructor is the only source of time complexity in a program. As the similar tick constructor in [9], it can represent different cost models and is more general than counting the number of reduction steps. For example, by adding `tick` after each input, we can count the number of communications in a process. By adding it after each replicated input on a channel a, we can count the number of calls to a. We can also count the number of reduction steps, by adding `tick` after each input and pattern matching.

As usual, the structural congruence $\equiv$ is defined as the least congruence containing:

$$P \mid 0 \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$

$$(\nu a)(\nu b)P \equiv (\nu b)(\nu a)P \qquad (\nu a)(P \mid Q) \equiv (\nu a)P \mid Q \text{ (when } a \text{ is not free in } Q)$$

Note that the last rule can always be applied from right to left by α-renaming. By associativity, we will often write parallel composition for any number of processes and not only two.

2.2 Parallel Complexity: The Span

We now review the definition of span [4]. We add a process construct $m : P$, where m is an integer. A process using this constructor will be called an *annotated process*. Intuitively, this annotated process has the meaning P *with* m *ticks before*. The congruence relation $\equiv$ is then enriched with the following relations:

$$0 : P \equiv P \qquad m : (P \mid Q) \equiv (m : P) \mid (m : Q)$$

$$m : (\nu a)P \equiv (\nu a)(m : P) \qquad m : (n : P) \equiv (m + n) : P$$

So, zero tick is equivalent to nothing and ticks can be distributed over parallel composition as expressed by the second relation. Name creation can be done before or after ticks without changing the semantics and finally ticks can be grouped together.

The rules for the reduction relation $\Rightarrow$ are given in Figure 1. This semantics works as the usual semantics for π-calculus, but when doing a synchronization, only the maximal annotation is kept, and ticks are memorized in the annotations. Span is then defined by:

▶ **Definition 1** (Parallel Complexity). *The* local complexity $\mathcal{C}_\ell(P)$ *of an annotated process* P *is defined by:*

$$\mathcal{C}_\ell(n : P) = n + \mathcal{C}_\ell(P) \qquad \mathcal{C}_\ell(P \mid Q) = \max(\mathcal{C}_\ell(P), \mathcal{C}_\ell(Q))$$

$$\mathcal{C}_\ell((\nu a)P) = \mathcal{C}_\ell(P) \qquad \mathcal{C}_\ell(P) = 0 \textit{ otherwise}$$

The global parallel complexity *(or* span*) of* P *is given by* $\max\{n \mid P \Rightarrow^* Q \wedge \mathcal{C}_\ell(Q) = n\}$ *where* $\Rightarrow^*$ *is the reflexive and transitive closure of* $\Rightarrow$*.*

▶ **Example 2.** Let $P := \mathtt{tick}.a().\mathtt{tick}.\overline{a}\langle\rangle \mid \mathtt{tick}.a().\mathtt{tick}.\overline{a}\langle\rangle \mid \overline{a}\langle\rangle$. Then, we have:

$$P \Rightarrow^2 1 : (a().\mathtt{tick}.\overline{a}\langle\rangle) \mid 1 : (a().\mathtt{tick}.\overline{a}\langle\rangle) \mid 0 : \overline{a}\langle\rangle \Rightarrow 1 : (a().\mathtt{tick}.\overline{a}\langle\rangle) \mid 1 : (\mathtt{tick}.\overline{a}\langle\rangle)$$
$$\Rightarrow 1 : (a().\mathtt{tick}.\overline{a}\langle\rangle) \mid 2 : \overline{a}\langle\rangle \Rightarrow 2 : (\mathtt{tick}.\overline{a}\langle\rangle) \Rightarrow 3 : \overline{a}\langle\rangle$$

Thus, the process has at least complexity 3. As all the other possible choices we could have made in the reduction steps are similar, the process has exactly complexity 3.

The following example motivates our introduction of usages in the next section.

▶ **Example 3** (Motivating Example). Let $P := a().\mathtt{tick}.\overline{a}\langle\rangle$. Then, the complexity of $P \mid P \mid P \mid \cdots \mid P \mid \overline{a}\langle\rangle$ is equal to the number of P in parallel.

3 Types with Usages

The goal of our work is to design a type system for processes such that if $\Gamma \vdash Q \triangleleft K$ then K is a bound on the complexity of Q, as in [4]. The analysis of [4] was not precise enough: in fact, the process P in Example 3 was not typable. The main idea to tackle this problem is to use the notion of usages to represent the channel-wise behaviour of processes. Usages have been used for deadlock-freedom analysis [23, 27, 30], but our notion of usages significantly differs from the original one, as discussed below.

3.1 Indices

First, we define integer indices, which are used to keep track of the size of values in a process.

▶ **Definition 4.** *Let* $\mathcal{V}$ *be a countable set of index variables, usually denoted by* i*,*j *or* k*. The set of* indices*, representing integers in* $\mathbb{N}_\infty = \mathbb{N} \cup \{\infty\}$*, is given by:*

$$I, J := I_{\mathbb{N}} \mid \infty \qquad I_{\mathbb{N}} := i \mid f(I_{\mathbb{N}}, \dots, I_{\mathbb{N}})$$

Here, $i \in \mathcal{V}$*. The symbol* f *is an element of a given set of function symbols containing, for example, integers constants as nullary operators, addition and multiplication. We also assume the subtraction as a function symbol, with* $n{-}m = 0$ *when* $m \geq n$*. Each function symbol* f *of arity* $\mathtt{ar}(f)$ *comes with an interpretation* $[\![f]\!] : \mathbb{N}^{\mathtt{ar}(f)} \to \mathbb{N}$*.*

Given an index valuation $\rho : \mathcal{V} \to \mathbb{N}$, we extend the interpretation of function symbols to indices, noted $[\![I]\!]_\rho$, as expected; $[\![I]\!]_\rho$ ranges over $\mathbb{N}_\infty$. For an index I, we write $I\{J_{\mathbb{N}}/i\}$ for the index obtained by replacing the occurrences of i in I with $J_{\mathbb{N}}$. Note that $\infty\{J_{\mathbb{N}}/i\} = \infty$.

▶ **Definition 5** (Constraints on Indices). *Let* $\varphi \subset \mathcal{V}$ *be a finite set of index variables. A* constraint C *on* φ *is an expression with the shape* $I \bowtie J$ *where* I *and* J *are indices with free variables in* φ *and* $\bowtie$ *denotes a binary relation on* $\mathbb{N}_\infty$*. Usually, we take* $\bowtie \in \{\leq, <, =, \neq\}$*. A finite set of constraints is denoted* Φ*.*

For a finite set $\varphi \subset \mathcal{V}$, we say that a valuation $\rho : \varphi \to \mathbb{N}$ *satisfies* a constraint $I \bowtie J$ on φ, noted $\rho \vDash I \bowtie J$ when $[\![I]\!]_\rho \bowtie [\![J]\!]_\rho$ holds. Similarly, $\rho \vDash \Phi$ holds when $\rho \vDash C$ for all $C \in \Phi$. Likewise, we note $\varphi; \Phi \vDash C$ when for all valuations ρ on φ such that $\rho \vDash \Phi$ we have $\rho \vDash C$. We will also use some extended operations on indices I, J; for example, we may use $\infty + J = \infty$ or other such equations.

3.2 Usages

We use *usages* to express the channel-wise behaviour of a process. Usages are a kind of CCS processes [29] on a single channel, where each action is annotated with two time intervals. The set of usages, ranged over by U and V, is given by:

$$U, V ::= 0 \mid (U|V) \mid \alpha^{A_o}_{J_c}.U \mid\ !U \mid U + V \qquad \alpha := \mathtt{In} \mid \mathtt{Out}$$

$$A_o, B_o ::= [I, J] \qquad J_c, I_c ::= J \mid [I, J]$$

Given a set of index variables φ and a set of constraints Φ, for an interval $[I, J]$, we always require that $\varphi; \Phi \vDash I \leq J$. For an interval $A_o = [I, J]$, we denote $\mathsf{Left}(A_o) = I$ and $\mathsf{Right}(A_o) = J$. In the original notion of usages [23, 27, 30], A_o and J_c were just numbers. The extension to intervals plays an important role in our analysis. Note that J_c may be a single index J, but this single index J should be understood as the interval $[-\infty, J]$.

Intuitively, a channel with usage 0 is not used at all. A channel of usage $U \mid V$ can be used according to U and V possibly in parallel. The usage $\mathtt{In}^{A_o}_{J_c}.U$ describes a channel that may be used for input, and then used according to U. The two intervals A_o and J_c, called *obligation* and *capacity* respectively, are used to achieve a kind of assume-guarantee reasoning. The obligation A_o indicates a *guarantee* that if the channel is indeed used for input, then the input should become ready during the interval A_o. The capacity J_c indicates the *assumption* that if the environment performs a corresponding output, that output will be provided during the time interval J_c after the input becomes ready. For example, if a channel a has usage $\mathtt{In}^{[1,1]}_{J_c}.0$, then the process $\mathtt{tick}.a().0$ conforms to the usage, but $a().0$ and $\mathtt{tick}.\mathtt{tick}.a().0$ do not. Similarly, $\mathtt{Out}^{A_o}_{J_c}.U$ has the same meaning but for output. The usage $!U$ denotes the usage U that can be replicated infinitely, and $U + V$ denotes a non-deterministic choice between the usages U and V. This is useful for example in a case of pattern matching where a channel can be used very differently in the two branches.

Recall that the obligation and capacity intervals in usages express a sort of assume-guarantee reasoning. We thus require that the assume-guarantee reasoning in a usage is "consistent" (or *reliable*, in the terminology of usages). For example, the usage $\mathtt{In}^{[0,0]}_{[1,1]} \mid \mathtt{Out}^{[1,1]}_0$ is reliable because (i) the part $\mathtt{In}^{[0,0]}_{[1,1]}$ assumes that a corresponding output will become ready at time 1, and the other part $\mathtt{Out}^{[1,1]}_0$ indeed guarantees that and moreover, (ii) $\mathtt{Out}^{[1,1]}_0$ assumes that a corresponding input will be ready by the time the output becomes ready, and the part $\mathtt{In}^{[0,0]}_{[1,1]}$ guarantees that. In contrast, the usage $\mathtt{In}^{[0,0]}_{[1,1]} \mid \mathtt{Out}^{[2,2]}_0$ is problematic because, although the part $\mathtt{In}^{[0,0]}_{[1,1]}$ assumes that an output will be ready at time 1, $\mathtt{Out}^{[2,2]}_0$ provides the output only at time 2. The consistency on assume-guarantee reasoning must hold during the whole computation; for example, in the usage $\mathtt{In}^{[0,0]}_{[0,0]}.\mathtt{In}^{[0,0]}_{[1,1]} \mid \mathtt{Out}^{[0,0]}_{[0,0]}.\mathtt{Out}^{[2,2]}_0$, the first input/output pair is fine, but the usage expressing the next communication: $\mathtt{In}^{[0,0]}_{[1,1]} \mid \mathtt{Out}^{[2,2]}_0$ is problematic. To properly define the reliability of usages during the whole computation, we first prepare a reduction semantics for usages, by viewing usages as CCS processes.

▶ **Definition 6** (Congruence for Usages). *The relation $\equiv$ is defined as the least congruence relation closed under:*

$$U \mid 0 \equiv U \qquad U \mid V \equiv V \mid U \qquad U \mid (V \mid W) \equiv (U \mid V) \mid W$$

$$!0 \equiv 0 \qquad !U \equiv !U \mid U \qquad !(U \mid V) \equiv !U \,|\, !V \qquad !!U \equiv !U$$

Before giving the reduction semantics, we introduce some notations.

▶ **Definition 7** (Operations on Usages). *We define the operations $\oplus$, $\sqcup$, and $+$ by:*

$$A_o \oplus J = [0, \mathsf{Left}(A_o) + J] \qquad A_o \oplus [I, J] = [\mathsf{Right}(A_o) + I, \mathsf{Left}(A_o) + J]$$
$$[I, J] \sqcup [I', J'] = [\max(I, I'), \max(J, J')] \qquad [I, J] + [I', J'] = [I + I', J + J']$$

Note that $\oplus$ is an operation that takes an obligation interval and a capacity and returns an interval. The operations $\sqcup$ (max) and $+$ are just pointwise extensions of the operations for indices. The intuition on $\oplus$ is explained later when we define the reduction relation.

The delaying operation $\uparrow^{A_o}U$ on usages is defined by:

$$\uparrow^{A_o}0 = 0 \qquad \uparrow^{A_o}(U \mid V) = \uparrow^{A_o}U \mid \uparrow^{A_o}V \qquad \uparrow^{A_o}(U + V) = \uparrow^{A_o}U + \uparrow^{A_o}V$$

$$\uparrow^{A_o}\alpha^{B_o}_{J_c}.U = \alpha^{A_o+B_o}_{J_c}.U \qquad \uparrow^{A_o}(!U) = !(\uparrow^{A_o}U)$$

We also define $[I, J]+J_c$ and thus $\uparrow^{J_c}U$ by extending the operation with: $[I, J]+J' = [I, J+J']$.

Intuitively, a usage $\uparrow^{A_o}U$ corresponds to the usage U delayed by a time approximated by the interval A_o.

The reduction relation is given by the rules of Figure 2. The first rule means that to reduce a usage, we choose one input and one output, and then we trigger the communication between them. This communication occurs and does not lead to an error when the capacity of an action indeed corresponds to a bound on the time the dual action is defined. This is given by the relation $A_o \subseteq B_o \oplus J_c$. As an example, let us suppose that $B_o = [1, 3]$, and the time for which the output becomes ready is in fact 2, then the capacity J_c says that after two units of time, the synchronization should happen in the interval J_c. So, if we take $J_c = [5, 7]$ for example, then if t is the time for which the dual input becomes ready, we must have $t \in [2 + 5, 2 + 7]$. This should be true for any time value in B_o, so we want that $\forall t' \in [1, 3], \forall t \in A_o, t \in [t' + 5, t' + 7]$, and this is equivalent to $A_o \subseteq B_o \oplus [5, 7] = [8, 8]$. Indeed, 8 is the only time that is in the three intervals $[6, 8]$, $[7, 9]$ and $[8, 10]$. The case where $J_c = J$ is a single index occurs when t can be smaller than t', and in this case we only ask that the upper bound is correct: $\forall t' \in B_o, \forall t \in A_o, t \leq t' + J$.

If the bound is incorrect, we trigger an error: see the second rule. In the case everything went well, the continuation is delayed by an approximation of the time when this communication occurs (see $\uparrow^{A_o \sqcup B_o}$ in the first rule). In the rules for $U + V$, a reduction step in usages can also make a non-deterministic choice.

An error in a usage reduction means that the assume-guarantee reasoning was inconsistent. Based on this intuition, we define the notion of reliablity.

▶ **Definition 8** (Reliability). *A usage U is* reliable *under $\varphi; \Phi$ if $U \not\longrightarrow^* \mathtt{err}$.*

▶ **Example 9.** Consider the usage $U := \mathtt{In}_1^{[1,1]}.\mathtt{Out}_0^{[1,1]} \mid \mathtt{In}_1^{[1,1]}.\mathtt{Out}_0^{[1,1]} \mid \mathtt{Out}_{[1,1]}^{[0,0]}$. The only possible reduction sequence (with symmetry) is:

$$U \longrightarrow \mathtt{Out}_0^{[2,2]} \mid \mathtt{In}_1^{[1,1]}.\mathtt{Out}_0^{[1,1]} \longrightarrow \mathtt{Out}_0^{[3,3]}.$$

$$\frac{\varphi;\Phi \vDash B_o \subseteq A_o \oplus I_c \qquad \varphi;\Phi \vDash A_o \subseteq B_o \oplus J_c}{\varphi;\Phi \vdash \mathtt{In}^{A_o}_{I_c}.U \mid \mathtt{Out}^{B_o}_{J_c}.V \longrightarrow {\uparrow}^{A_o \sqcup B_o}(U \mid V)} \qquad \frac{\varphi;\Phi \nvDash (B_o \subseteq A_o \oplus I_c \wedge A_o \subseteq B_o \oplus J_c)}{\varphi;\Phi \vdash \mathtt{In}^{A_o}_{I_c}.U \mid \mathtt{Out}^{B_o}_{J_c}.V \longrightarrow \mathtt{err}}$$

$$\frac{}{\varphi;\Phi \vdash U + V \longrightarrow U} \qquad \frac{}{\varphi;\Phi \vdash U + V \longrightarrow V} \qquad \frac{\varphi;\Phi \vdash U \longrightarrow U' \qquad U' \neq \mathtt{err}}{\varphi;\Phi \vdash U \mid V \longrightarrow U' \mid V}$$

$$\frac{\varphi;\Phi \vdash U \longrightarrow \mathtt{err}}{\varphi;\Phi \vdash U \mid V \longrightarrow \mathtt{err}} \qquad \frac{U \equiv U' \qquad \varphi;\Phi \vdash U' \longrightarrow V' \qquad V' \equiv V}{\varphi;\Phi \vdash U \longrightarrow V}$$

Figure 2 Reduction Rules for Usages.

For the first step, we have indeed $[1,1] \subseteq [0,0] \oplus [1,1] = [1,1]$ and $[0,0] \subseteq [1,1] \oplus 1 = [0,2]$. Note that the capacity $[0,1]$ instead of 1 for the input would not have worked since $[1,1] \oplus [0,1] = [1,2]$. Thus, the usage U is reliable. It corresponds, for example, to the usage of the channel a in the process P given in Example 2: $\mathtt{tick}.a().\mathtt{tick}.\overline{a}\langle\rangle \mid \mathtt{tick}.a().\mathtt{tick}.\overline{a}\langle\rangle \mid \overline{a}\langle\rangle$. The obligation $[1,1]$ corresponds to waiting for exactly one tick. Then, the capacities say that once they are ready, the two inputs will indeed communicate before one time unit for any reduction. And at the end, we obtain an output available at time 3, and this output has no communication. One can see that those capacities and obligations indeed give the complexity of this process. Thus, we will ask in the type system that all usages are reliable, and so the time indications will give some complexity bounds on the behaviour of a channel. ⌟

▶ **Example 10.** We give an example of a non-reliable usage. To the previous example, let us add another input in parallel

$$U := \mathtt{In}^{[1,1]}_{1}.\mathtt{Out}^{[1,1]}_{0} \mid \mathtt{In}^{[1,1]}_{1}.\mathtt{Out}^{[1,1]}_{0} \mid \mathtt{In}^{[1,1]}_{1}.\mathtt{Out}^{[1,1]}_{0} \mid \mathtt{Out}^{[0,0]}_{[1,1]}$$

We have: $U \longrightarrow^* \mathtt{Out}^{[3,3]}_{0} \mid \mathtt{In}^{[1,1]}_{1}.\mathtt{Out}^{[1,1]}_{0} \longrightarrow \mathtt{err}$, because $[1,1] \oplus 1 = [0,2]$, so the capacity here is not a good assumption. However, the following variant of the usage:

$$U := \mathtt{In}^{[1,1]}_{2}.\mathtt{Out}^{[1,1]}_{0} \mid \mathtt{In}^{[1,1]}_{2}.\mathtt{Out}^{[1,1]}_{0} \mid \mathtt{In}^{[1,1]}_{2}.\mathtt{Out}^{[1,1]}_{0} \mid \mathtt{Out}^{[0,0]}_{[1,1]}$$

is reliable. This example shows how reliability adapts to parallel composition. ⌟

We introduce another relation $U \sqsubseteq V$ called the *subusage* relation, which will be used later to define the subtyping relation. It is defined by the rules of Figure 3. The relation $U \sqsubseteq V$ intuitively means that any channel of usage U may also be used according to V. For example, $U \sqsubseteq 0$ says that we may not use a channel (usage equal to 0). Recall that an obligation and a capacity express a guarantee and an assumption respectively. The last but one rule says that it is safe to strengthen the guarantee and weaken the assumption. We use the relation $I_c \leq J_c$ to denote the relation $\subseteq$ on intervals, where a single index J is considered as the interval $[-\infty, J]$. The last rule can be understood as follows. The part ${\uparrow}^{A_o + J_c} V$ says that a channel may be used according to V only after the interval $A_o + J_c$. Since the action $\alpha^{A_o}_{J_c}$ is indeed finished during the interval $A_o + J_c$, we can move V to under the guard of $\alpha^{A_o}_{J_c}$. This last rule is especially useful for substitution, as explained in the example below.

▶ **Example 11.** Consider the process:

$$P := a(r).r().b() \mid \overline{a}\langle b\rangle$$

Let us give usages to b and r; here we omit time annotations for the sake of simplicity. We have $U_r = \mathtt{In}$ and $U_b = \mathtt{In} \mid U_r$ Indeed, r is used only once as an input, and b is used as an input on the left, and it is sent to be used as r on the right. Thus, after a reduction step we

$$\frac{}{\varphi;\Phi \vdash U \sqsubseteq 0} \qquad \frac{i \in \{1;2\}}{\varphi;\Phi \vdash U_1 + U_2 \sqsubseteq U_i} \qquad \frac{\varphi;\Phi \vdash U \sqsubseteq U'}{\varphi;\Phi \vdash U + V \sqsubseteq U' + V}$$

$$\frac{\varphi;\Phi \vdash V \sqsubseteq V'}{\varphi;\Phi \vdash U + V \sqsubseteq U + V'} \qquad \frac{\varphi;\Phi \vdash U \sqsubseteq U'}{\varphi;\Phi \vdash U \mid V \sqsubseteq U' \mid V} \qquad \frac{\varphi;\Phi \vdash U \sqsubseteq U'}{\varphi;\Phi \vdash !U \sqsubseteq !U'}$$

$$\frac{U \equiv U' \qquad \varphi;\Phi \vdash U' \sqsubseteq V' \qquad V \equiv V'}{\varphi;\Phi \vdash U \sqsubseteq V} \qquad \frac{\varphi;\Phi \vdash U \sqsubseteq U' \qquad \varphi;\Phi \vdash U' \sqsubseteq U''}{\varphi;\Phi \vdash U \sqsubseteq U''}$$

$$\frac{\varphi;\Phi \vdash U \sqsubseteq U'}{\varphi;\Phi \vdash \alpha^{A_o}_{J_c}.U \sqsubseteq \alpha^{A_o}_{J_c}.U'} \qquad \frac{\varphi;\Phi \vDash B_o \subseteq A_o \qquad \varphi;\Phi \vDash I_c \leq J_c}{\varphi;\Phi \vdash \alpha^{A_o}_{I_c}.U \sqsubseteq \alpha^{B_o}_{J_c}.U}$$

$$\frac{}{\varphi;\Phi \vdash (\alpha^{A_o}_{J_c}.U) \mid (\uparrow^{A_o+J_c} V) \sqsubseteq \alpha^{A_o}_{J_c}.(U \mid V)}$$

Figure 3 Subusage.

$$\frac{\varphi;\Phi \vDash I' \leq I \qquad \varphi;\Phi \vDash J \leq J'}{\varphi;\Phi \vdash \mathsf{Nat}[I,J] \sqsubseteq \mathsf{Nat}[I',J']} \qquad \frac{\varphi;\Phi \vdash \widetilde{T} \sqsubseteq \widetilde{T'} \qquad \varphi;\Phi \vdash \widetilde{T'} \sqsubseteq \widetilde{T} \qquad \varphi;\Phi \vdash U \sqsubseteq V}{\varphi;\Phi \vdash \mathtt{ch}(\widetilde{T})/U \sqsubseteq \mathtt{ch}(\widetilde{T'})/V}$$

$$\frac{\varphi,\widetilde{i};\Phi \vdash \widetilde{T} \sqsubseteq \widetilde{T'} \qquad \varphi,\widetilde{i};\Phi \vdash \widetilde{T'} \sqsubseteq \widetilde{T} \qquad \varphi,\widetilde{i};\Phi \vDash K = K' \qquad \varphi;\Phi \vdash U \sqsubseteq V}{\varphi;\Phi \vdash \forall\widetilde{i}.\mathtt{srv}^K(\widetilde{T})/U \sqsubseteq \forall\widetilde{i}.\mathtt{srv}^{K'}(\widetilde{T'})/V}$$

Figure 4 Subtyping Rules for Usage Types.

obtain $P \to b().b()$ where b has usage $U'_b = \mathtt{In}.\mathtt{In}$. So, the channel b had usage U_b in P, but it ended up being used according to U'_b; that is valid since we have the subusage relation $U_b \sqsubseteq U'_b$.

3.3 Type System

We extend ordinary types for the π-calculus with usages.

▶ **Definition 12** (Usage Types). *We define* types *by the following grammar:*

$$T, S ::= \mathsf{Nat}[I,J] \mid \mathtt{ch}(\widetilde{T})/U \mid \forall\widetilde{i}.\mathtt{srv}^K(\widetilde{T})/U.$$

The type $\mathsf{Nat}[I,J]$ describes an integer n such that $I \leq n \leq J$. Channels are classified into *server channels* (or just *servers*) and *simple channels*. All the inputs on a server channel must be replicated (as in $!a(\widetilde{v}).P$), while no input on a simple channel can be replicated. The type $\mathtt{ch}(\widetilde{T})/U$ describes a simple channel that is used for transmitting values of type $\widetilde{T}$ according to usage U. For example, $\mathtt{ch}(\mathsf{Nat}[I,J])/U$ is the type of channels used according to U for transmitting integers in the interval $[I,J]$. The type $\forall\widetilde{i}.\mathtt{srv}^K(\widetilde{T})/U$ describes a server channel that is used for transmitting values of type $\widetilde{T}$ according to usage U; the superscript K, which we call the *complexity* of a server, is an interval. It denotes the cost incurred when a server is invoked. Note that the server type allows polymorphism on index variables $\widetilde{i}$.

The subtyping relation $T \sqsubseteq T'$, which means that a value of type T can also be used as a value of type T', is defined by the rules of Figure 4.

We extend operations on usages to partial operations on types and typing contexts with $\Gamma = v_1 : T_1, \ldots, v_n : T_n$. The delaying of a type $\uparrow^{A_o} T$ is defined as the delaying of the usage for a channel or a server type, and it does nothing on integers. We also say that a type is *reliable* when it is an integer type, or when it is a server or channel type with a reliable usage. We define following operations:

$$\frac{v : T \in \Gamma}{\varphi; \Phi; \Gamma \vdash v : T} \qquad \frac{}{\varphi; \Phi; \Gamma \vdash 0 : \mathsf{Nat}[0,0]} \qquad \frac{\varphi; \Phi; \Gamma \vdash e : \mathsf{Nat}[I,J]}{\varphi; \Phi; \Gamma \vdash \mathsf{s}(e) : \mathsf{Nat}[I+1, J+1]}$$

$$\frac{\varphi; \Phi; \Delta \vdash e : T' \qquad \varphi; \Phi \vdash \Gamma \sqsubseteq \Delta \qquad \varphi; \Phi \vdash T' \sqsubseteq T}{\varphi; \Phi; \Gamma \vdash e : T}$$

Figure 5 Typing Rules for Expressions.

▶ **Definition 13.** *The parallel composition $T \,|\, T'$ is defined by:*

$$\mathsf{Nat}[I,J] \mid \mathsf{Nat}[I,J] = \mathsf{Nat}[I,J] \qquad \mathit{ch}(\widetilde{T})/U \mid \mathit{ch}(\widetilde{T})/V = \mathit{ch}(\widetilde{T})/(U \mid V)$$

$$\forall \widetilde{i}.\mathit{srv}^K(\widetilde{T})/U \mid \forall \widetilde{i}.\mathit{srv}^K(\widetilde{T})/V = \forall \widetilde{i}.\mathit{srv}^K(\widetilde{T})/(U \mid V)$$

▶ **Definition 14** (Replication of Type)**.** *The replication of a type $!T$ is defined by:*

$$!\mathsf{Nat}[I,J] = \mathsf{Nat}[I,J] \qquad !\mathit{ch}(\widetilde{T})/U = \mathit{ch}(\widetilde{T})/(!U) \qquad !\forall \widetilde{i}.\mathit{srv}^K(\widetilde{T})/U = \forall \widetilde{i}.\mathit{srv}^K(\widetilde{T})/(!U)$$

The (partial) operations on types defined above are extended pointwise to contexts. For example, for $\Gamma = v_1 : T_1, \ldots, v_n : T_n$ and $\Delta = v_1 : T'_1, \ldots, v_n : T'_n$, we define $\Gamma \mid \Delta = v_1 : T_1 \mid T'_1, \ldots, v_n : T_n \mid T'_n$. Note that this is defined just if Γ and Δ agree on the typing of integers and associate the same types (excluding usage) to names.

▶ **Definition 15.** *Given a capacity J_c and an interval $K = [K_1, K_2]$, we define $J_c; K$ by;*

$$J; [K_1, K_2] = [0, J + K_2] \qquad [\infty, \infty]; [K_1, K_2] = [0, 0] \qquad [I_{\mathbb{N}}, J]; [K_1, K_2] = [0, J + K_2]$$

Intuitively, $J_c; K$ represents the complexity of an input/output process when the input/output has capacity J_c and the complexity of the continuation is K. $J_c = [\infty, \infty]$ means the input/output will never succeed (because there is no corresponding output/input); hence the complexity is 0. A case where this is useful is given later in Example 22. Otherwise, an upper-bound is given by $J + K_2$ (the time spent for the input/output to succeed, plus K_2). The lower-bound is 0, since the input/output may be blocked forever.

The type system is given in Figures 5 and 6. The typing rules for expressions are standard ones for sized types.

A type judgment for processes is of the form $\varphi; \Phi; \Gamma \vdash P \triangleleft [I, J]$ where φ denotes the set of index variables, Φ is a set of constraints on index variables, and J is a bound on the parallel complexity of P under those constraints. This complexity bound J can also be seen as a bound on the open complexity of a process, that is to say the complexity of P in an environment corresponding to the types in Γ. For example, a channel with usage $\mathtt{In}_5^{[1,1]}$ alone cannot be reduced, as it is only used as an input. So, the typing $\cdot; \cdot; a : \mathtt{ch}()/\mathtt{In}_5^{[1,1]} \vdash \mathtt{tick}.a() \triangleleft [1, 6]$ says that in an environment that may provide an output on the channel a within the time interval $[1, 1] \oplus 5 = [0, 6]$, this process has a complexity bounded by 6. Similarly, the lower bound I is a lower bound on the parallel complexity of P. But in practice, this lower bound is often too imprecise.[1]

The (par) rule separates a context into two parts, and the complexity is the maximum over the two complexities, both for lower bound and upper bound. The (tick) rule shows the addition of a tick implies a delay of $[1, 1]$ in the context and the complexity. The (nu)

[1] This is because in the definition of $J_C; K$ in Definition 15, we pessimistically take into account the possibility that each input/output may be blocked forever. We can avoid the pessimistic estimation of the lower-bound by incorporating information about lock-freedom [23, 25].

$$\text{(zero)}\ \frac{}{\varphi;\Phi;\Gamma\vdash 0\triangleleft[0,0]}\qquad \text{(par)}\ \frac{\varphi;\Phi;\Gamma\vdash P\triangleleft K_1\qquad \varphi;\Phi;\Delta\vdash Q\triangleleft K_2}{\varphi;\Phi;\Gamma\mid\Delta\vdash P\mid Q\triangleleft K_1\sqcup K_2}$$

$$\text{(tick)}\ \frac{\varphi;\Phi;\Gamma\vdash P\triangleleft K}{\varphi;\Phi;\uparrow^{[1,1]}\Gamma\vdash \texttt{tick}.P\triangleleft K+[1,1]}\qquad \text{(ich)}\ \frac{\varphi;\Phi;\Gamma,a:\texttt{ch}(\widetilde{T})/U,\widetilde{v}:\widetilde{T}\vdash P\triangleleft K}{\varphi;\Phi;\uparrow^{J_c}\Gamma,a:\texttt{ch}(\widetilde{T})/\texttt{In}^{[0,0]}_{J_c}.U\vdash a(\widetilde{v}).P\triangleleft J_c;K}$$

$$\text{(iserv)}\ \frac{(\varphi,\widetilde{i});\Phi;\Gamma,a:\forall\widetilde{i}.\texttt{srv}^K(\widetilde{T})/U,\widetilde{v}:\widetilde{T}\vdash P\triangleleft K}{\varphi;\Phi;\uparrow^{J_c}!\Gamma,a:\forall\widetilde{i}.\texttt{srv}^K(\widetilde{T})/!\texttt{In}^{[0,0]}_{J_c}.U\vdash !a(\widetilde{v}).P\triangleleft[0,0]}$$

$$\text{(och)}\ \frac{\varphi;\Phi;\Gamma',a:\texttt{ch}(\widetilde{T})/V\vdash\widetilde{e}:\widetilde{T}\qquad \varphi;\Phi;\Gamma,a:\texttt{ch}(\widetilde{T})/U\vdash P\triangleleft K}{\varphi;\Phi;\uparrow^{J_c}(\Gamma\mid\Gamma'),a:\texttt{ch}(\widetilde{T})/\texttt{Out}^{[0,0]}_{J_c}.(V\mid U)\vdash\overline{a}\langle\widetilde{e}\rangle.P\triangleleft J_c;K}$$

$$\text{(oserv)}\ \frac{\varphi;\Phi;\Gamma',a:\forall\widetilde{i}.\texttt{srv}^K(\widetilde{T})/V\vdash\widetilde{e}:\widetilde{T}\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\}\qquad \varphi;\Phi;\Gamma,a:\forall\widetilde{i}.\texttt{srv}^K(\widetilde{T})/U\vdash P\triangleleft K'}{\varphi;\Phi;\uparrow^{J_c}(\Gamma\mid\Gamma'),a:\forall\widetilde{i}.\texttt{srv}^K(\widetilde{T})/\texttt{Out}^{[0,0]}_{J_c}.(V\mid U)\vdash\overline{a}\langle\widetilde{e}\rangle.P\triangleleft J_c;(K'\sqcup K\{\widetilde{I_{\mathbb{N}}}/\widetilde{i}\})}$$

$$\text{(if)}\ \frac{\varphi;\Phi;\Gamma\vdash e:\mathsf{Nat}[I,J]\qquad \varphi;\Phi,I\leq 0;\Gamma\vdash P\triangleleft K\qquad \varphi;\Phi,J\geq 1;\Gamma,x:\mathsf{Nat}[I{-}1,J{-}1]\vdash Q\triangleleft K}{\varphi;\Phi;\Gamma\vdash\texttt{match}\ e\ \{\texttt{case 0}\mapsto P;\texttt{case s}(x)\mapsto Q\}\triangleleft K}$$

$$\text{(nu)}\ \frac{\varphi;\Phi;\Gamma,a:T\vdash P\triangleleft K\qquad T\text{ reliable}}{\varphi;\Phi;\Gamma\vdash(\nu a)P\triangleleft K}$$

$$\text{(subtype)}\ \frac{\varphi;\Phi;\Delta\vdash P\triangleleft K\qquad \varphi;\Phi\vdash\Gamma\sqsubseteq\Delta\qquad \varphi;\Phi\vDash K\subseteq K'}{\varphi;\Phi;\Gamma\vdash P\triangleleft K'}$$

Figure 6 Typing Rules for Processes.

rule imposes that all names must have a reliable usage when they are created. In order to type a channel with the (ich) rule, the channel must have an input usage, with obligation $[0,0]$. Note that with the subusage relation, we have $\texttt{In}^{A_o}_{J_c}\sqsubseteq\texttt{In}^{[0,0]}_{J_c}$ if and only if $A_o=[0,I]$ for some I. So, this typing rule imposes that the lower-bound guarantee is correct, but the rule is not restrictive for upper-bound. This rule induces a delay of J_c in both context and complexity. Indeed, in practice this input does not happen immediately as we need to wait for output. This is where the assumption on when this output is ready, given by the capacity, is useful. The rule for output (och) is similar. For a server, the rule for input (iserv) is similar to (ich) in principle but differs in the way complexity is managed. Indeed, as a replicated input is never modified nor erased through a computation, giving it a non-zero complexity would harm the precision of the type system. Moreover, if this server represents for example a function on an integer with linear complexity, then the complexity of this server depends on the size of the integer it receives; that is why the complexity is transferred to the output rule on server, as one can see in the rule (oserv). Indeed, this rule (oserv) is again similar to (och) but the complexity of a call to the server is added in the rule. As we have polymorphism on servers, in order to type an output we need to find an instantiation on the indices $\widetilde{i}$, which is denoted by $\widetilde{I_{\mathbb{N}}}$ in this rule. Finally, the (if) rule is the only rule that modifies the set of constraints, and it gives information on the values the sizes can take. As explained in Example 21, those constraints are crucial in our sized type system. Note that contexts are not separated in this rule. So, for both branches, it means that the usage of channels must be the same. However, because we have the choice usage $(U+V)$, in practice we can use different usages in those two branches.

▶ **Example 16.** The typing derivation of the process in Example 2 is given in Figure 7. Note that the process $(\texttt{tick}.a().\texttt{tick}.\overline{a}\langle\rangle\mid\texttt{tick}.a().\texttt{tick}.\overline{a}\langle\rangle\mid\texttt{tick}.a().\texttt{tick}.\overline{a}\langle\rangle\mid\overline{a}\langle\rangle$ is also typable, in the same way, using the following usage (recall Example 10).

$$U:=\texttt{In}^{[1,1]}_2.\texttt{Out}^{[1,1]}_0\mid\texttt{In}^{[1,1]}_2.\texttt{Out}^{[1,1]}_0\mid\texttt{In}^{[1,1]}_2.\texttt{Out}^{[1,1]}_0\mid\texttt{Out}^{[0,0]}_{[1,1]}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot;\cdot;a : \texttt{ch}()/\texttt{Out}_0^{[0,0]} \vdash \overline{a}\langle\rangle \triangleleft [0,0]}}{\cdot;\cdot;a : \texttt{ch}()/\texttt{Out}_0^{[1,1]} \vdash \texttt{tick}.\overline{a}\langle\rangle \triangleleft [1,1]}}{\cdot;\cdot;a : \texttt{ch}()/(\texttt{In}_1^{[0,0]}.\texttt{Out}_0^{[1,1]}) \vdash a().\texttt{tick}.\overline{a}\langle\rangle \triangleleft [0,2]}}{\cdot;\cdot;a : \texttt{ch}()/(\texttt{In}_1^{[1,1]}.\texttt{Out}_0^{[1,1]}) \vdash \texttt{tick}.a().\texttt{tick}.\overline{a}\langle\rangle \triangleleft [1,3]} \qquad \dfrac{}{\cdot;\cdot;a : \texttt{ch}()/(\texttt{Out}_{[1,1]}^{[0,0]}) \vdash \overline{a}\langle\rangle \triangleleft [0,1]}}{\cdot;\cdot;a : \texttt{ch}()/(\texttt{In}_1^{[1,1]}.\texttt{Out}_0^{[1,1]} \mid \texttt{In}_1^{[1,1]}.\texttt{Out}_0^{[1,1]} \mid \texttt{Out}_{[1,1]}^{[0,0]}) \vdash \texttt{tick}.a().\texttt{tick}.\overline{a}\langle\rangle \mid \cdots \mid \overline{a}\langle\rangle \triangleleft [1,3]}$$

Figure 7 Typing of Example 2.

We thus obtain the complexity bound $[1,4]$.

An example for the use of servers and sizes is given later, in Example 21, as well as a justification for the use of intervals for obligations and capacities, in Example 20.

4 Soundness and Examples

The proof of soundness relies on subject reduction. In order to work on the parallel reduction relation $\Rightarrow$, we need to consider annotated processes. We introduce the following typing rule, for the annotation, as a generalization of the rule for `tick`.

$$\dfrac{\varphi;\Phi;\Gamma \vdash P \triangleleft K}{\varphi;\Phi;\uparrow^{[m,m]}\Gamma \vdash m : P \triangleleft K + [m,m]}$$

4.1 Subject Reduction and Soundness

In the Appendix B.2, we describe some intermediate lemmas needed for the soundness proof, namely weakening, strengthening and then substitution lemmas for index and expressions.

Let us first introduce a notation for subject reduction:

▶ **Definition 17** (Reduction for Contexts). *We say that a context Γ reduces to a context Γ' under $\varphi;\Phi$, denoted $\varphi;\Phi \vdash \Gamma \longrightarrow^* \Gamma'$ when one of the following holds:*

- $\Gamma = \Delta, a : \mathit{ch}(\widetilde{T})/U \qquad \varphi;\Phi \vdash U \longrightarrow^* U' \qquad \Gamma' = \Delta, a : \mathit{ch}(\widetilde{T})/U'$
- $\Gamma = \Delta, a : \forall\widetilde{i}.\mathit{srv}^K(\widetilde{T})/U \qquad \varphi;\Phi \vdash U \longrightarrow^* U' \qquad \Gamma' = \Delta, a : \forall\widetilde{i}.\mathit{srv}^K(\widetilde{T})/U'$

So, Γ' is Γ after some reduction steps but only in a unique usage. We obtain immediately that if all types in Γ are reliable then all types in Γ' are also reliable by definition of reliability.

The subject reduction property is stated as follows; see Appendix B.5 for a proof.

▶ **Theorem 18** (Subject Reduction). *If $\varphi;\Phi;\Gamma \vdash P \triangleleft K$ with all types in Γ reliable and $P \Rightarrow Q$ then there exists Γ' with $\varphi;\Phi \vdash \Gamma \longrightarrow^* \Gamma'$ and $\varphi;\Phi;\Gamma' \vdash Q \triangleleft K$.*

The following is the main soundness theorem.

▶ **Theorem 19.** *Let P be an annotated process and n be its global parallel complexity. Then, if $\varphi;\Phi;\Gamma \vdash P \triangleleft [I,J]$ with all types in Γ reliable, then we have $\varphi;\Phi \vDash J \geq n$. Moreover, if Γ does not contain any integers variables, we have $\varphi;\Phi \vDash I \leq n$.*

Proof. By Theorem 18, all reductions from P using $\Rightarrow$ conserve the typing. The context may be reduced too, but as a reduction step does not harm reliability, we can still apply the subject reduction through all the reduction steps of $\Rightarrow$. Moreover, for a process Q, if we have a typing $\varphi;\Phi;\Gamma \vdash Q \triangleleft [I,J]$, then $J \geq \mathcal{C}_\ell(Q)$. Thus, J is indeed a bound on the parallel complexity by definition. As for the lower bound, one can see that we do not always have $I \leq \mathcal{C}_\ell(Q)$ because of the processes $\mathtt{tick}.Q'$ and $\mathtt{match}\ e\ \{\mathtt{case}\ 0 \mapsto Q_1; \mathtt{case}\ \mathtt{s}(x) \mapsto Q_2\}$. However, those two processes are not in normal form for $\Rightarrow$, because $\mathtt{tick}.Q' \Rightarrow 1 : Q'$ and as there are no integer variables in Γ, the pattern matching can also be reduced. Thus, from a process Q we can find Q' such that $Q \Rightarrow Q'$ and Q' has no such processes of this shape on the top. And then, we obtain $I \leq \mathcal{C}_\ell(Q')$ which is smaller than the parallel complexity of Q by definition. ◀

4.2 Examples

Below we give several examples to demonstrate the expressive power of our type system.

▶ **Example 20** (Intervals). To see the need for an interval capacity, consider the following process:

$$a().\overline{b}\langle\rangle \mid \mathtt{match}\ e\ \{\mathtt{case}\ 0 \mapsto \overline{a}\langle\rangle; \mathtt{case}\ \mathtt{s}(x) \mapsto \mathtt{tick}.\overline{a}\langle\rangle\}$$

Depending on the value of e (which may be statically unknown), an output on a may be available at time 0 or 1. Thus, the input usage on a should have a capacity interval $[0, 1]$. As a result, the obligation of the output usage on b should also be an interval $[0, 1]$.

Now, one may think that we can assume that lower-bounds are always 0 and omit lower-bounds, since we are mainly interested in an *upper-bound* of the parallel complexity. Information about lower-bounds is, however, actually required for precise reasoning on upper-bounds. For example, consider the process

$$a().\overline{b}\langle\rangle \mid \mathtt{tick}.\overline{a}\langle\rangle.b()$$

With intervals, a have the usage $\mathtt{In}^{[0,0]}_{[1,1]} \mid \mathtt{Out}^{[1,1]}_{0}$ and so b has the usage $\mathtt{Out}^{[1,1]}_{[0,0]} \mid \mathtt{In}^{[1,1]}_{[0,0]}$, and the parallel complexity of the process can be precisely inferred to be 1.

If we set lower-bounds to 0 and assign to a the usage $\mathtt{In}^{[0,0]}_{[0,1]} \mid \mathtt{Out}^{[0,1]}_{0}$, then the usage of b can only be: $\mathtt{Out}^{[0,1]}_{1} \mid \mathtt{In}^{[0,1]}_{1}$. Note that according to the imprecise usage of a, the output on b may become ready at time 0 and then have to wait for one time unit until the input on b becomes ready; thus, the capacity of the output on b is 1, instead of $[0, 0]$. An upper-bound of the parallel complexity would therefore be inferred to be $1 + 1 = 2$ (because the usages tell us that the lefthand side process may wait for one time unit at a, and then for another time unit at b), which is too imprecise.

We remark that this problem does not come for an inappropriate definitions of usages with only upper-bound in our work. Indeed, by adapting the usage type system given in [23], we would have the same imprecision. In the same way, trying to give a notion of reliability that makes the usage $\mathtt{Out}^{[0,1]}_{0} \mid \mathtt{In}^{[0,1]}_{0}$ reliable would lead to an unsound type system, as it would make the subusage relation less flexible, which is essential for soundness. ⌟

Let us also present how sizes and polymorphism over indices in servers can type processes defined by replication such as the factorial. Please note that by taking inspiration from the typing in [4], using the type representation given in the Appendix A, more complicated examples of parallel programs such as the bitonic sort could be typed in our setting with a good complexity bound.

▶ **Example 21** (Factorial). Assume a function on expressions $\mathtt{mult} : \mathsf{Nat}[I, J] \times \mathsf{Nat}[I', J'] \to \mathsf{Nat}[I * I', J * J']$. In practice, this should be encoded as a server in π-calculus, but for simplicity, we consider it as a function. We will describe the factorial and count the number

$$P := !f(n,r).\mathtt{match}\ n\ \{\mathtt{case}\ \mathtt{0} \mapsto \overline{r}\langle 1\rangle; \mathtt{case}\ \mathtt{s}(m) \mapsto (\nu r')(\overline{f}\langle m, r'\rangle \mid r'(x).\mathtt{tick}.\overline{r}\langle \mathit{mult}(n,x)\rangle)\}$$

$$\dfrac{\dfrac{\dfrac{}{i;\cdot;n:\mathsf{Nat}[i]\vdash n:\mathsf{Nat}[i]} \qquad \dfrac{i;(i\le 0)\vDash i!=1}{\overline{i;i\le 0;f:T',n:\mathsf{Nat}[i],r:\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]}\vdash \overline{r}\langle 1\rangle \triangleleft [0,i]}} \qquad \pi_1}{i;\cdot;f:T',n:\mathsf{Nat}[i],r:\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]}\vdash \mathtt{match}\ n\ \{\mathtt{case}\ \mathtt{0}\mapsto \overline{r}\langle 1\rangle;\mathtt{case}\ \mathtt{s}(m)\mapsto \cdots\}\triangleleft[0,i]}}{\cdot;\cdot;f:T\vdash !f(n,r).\mathtt{match}\ n\ \{\mathtt{case}\ \mathtt{0}\mapsto \overline{r}\langle 1\rangle;\mathtt{case}\ \mathtt{s}(m)\mapsto \cdots\}\triangleleft[0,0]}$$

with the main branch of π_1 being:

$$\dfrac{\dfrac{\cdots \qquad \dfrac{\dfrac{\dfrac{\dfrac{(i;i\ge 1)\vDash i*(i{-}1)!=i!}{\overline{i;i\ge 1;n:\mathsf{Nat}[i],x:\mathsf{Nat}[(i{-}1)!]\vdash \mathit{mult}(n,x):\mathsf{Nat}[i!]}}}{i;i\ge 1;n:\mathsf{Nat}[i],x:\mathsf{Nat}[(i{-}1)!],r:\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[0,0]}\vdash \overline{r}\langle \mathit{mult}(n,x)\rangle\triangleleft[0,0]}}{i;i\ge 1;n:\mathsf{Nat}[i],x:\mathsf{Nat}[(i{-}1)!],r:\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[1,1]}\vdash \mathtt{tick}.\overline{r}\langle \mathit{mult}(n,x)\rangle\triangleleft[1,1]}}{i;i\ge 1;n:\mathsf{Nat}[i],r':S_2,r:\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]}\vdash r'(x).\cdots\triangleleft[0,i]}}{i;i\ge 1;n:\mathsf{Nat}[i],m:\mathsf{Nat}[i{-}1],r:\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]},f:T',r':S\vdash \overline{f}\langle m,r'\rangle \mid r'(x).\cdots\triangleleft[0,i]}}{i;i\ge 1;n:\mathsf{Nat}[i],m:\mathsf{Nat}[i{-}1],r:\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]},f:T'\vdash(\nu r')\cdots\triangleleft[0,i]}$$

Figure 8 Representation and Typing of Factorial.

of multiplications with tick. We write $\mathsf{Nat}[I]$ to denote $\mathsf{Nat}[I,I]$. We use the usual notation $I!$ to represent the factorial function in indices. The process representing factorial and its typing derivation are given in Figure 8. The following type T denotes:

$$\forall i.\mathtt{srv}^{[0,i]}(\mathsf{Nat}[i],\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]})/(!\mathtt{In}_\infty^{[0,0]}.\mathtt{Out}_0^{[0,\infty]})$$

Denoting a server taking an integer as input, and a return channel on which the factorial of this integer is sent, in i units of time. The usage of this server describes that it can be called anytime. This type is reliable and it would be reliable even if composed with any kind of output $\mathtt{Out}_0^{A_o}$ if we want to call this server. Let:

$$T' = \forall i.\mathtt{srv}^{[0,i]}(\mathsf{Nat}[i],\mathtt{ch}(\mathsf{Nat}[i!])/\mathtt{Out}_0^{[i,i]})/\mathtt{Out}_0^{[0,\infty]})$$

$$S = \mathtt{ch}(\mathsf{Nat}[(i{-}1)!])/(\mathtt{Out}_0^{[i-1,i-1]} \mid \mathtt{In}_{[i-1,i-1]}^{[0,0]}) = S_1 \mid S_2$$

where S_1 and S_2 are obtained by the expected separation of the usage. This type S is reliable under $(i);(i \geq 1)$. Thus, we give the typing described in Figure 8. From the type of f, we see on its complexity $[0,i]$ that it does at most a linear number of multiplications. Note that the constraints that appear in a match are useful since without them, we could not prove $i;(i \leq 0) \vDash i! = 1$ and $i;(i \geq 1) \vDash i*(i{-}1)! = i!$. Moreover, polymorphism over indices is necessary in order to find that the recursive call is made on a strictly smaller size $i{-}1$. ⌟

Let us now justify the use of this operator $J_c; K$ in order to treat complexity.

▶ **Example 22** (Deadlock). Let us consider the process $P := (\nu a)(\nu b)(a().\mathtt{tick}.\overline{b}\langle\rangle \mid b().\mathtt{tick}.\overline{a}\langle\rangle)$. P is typed as shown in Figure 9. As a and b have exactly the same behaviour, let us focus on the typing of $a().\mathtt{tick}.\overline{b}\langle\rangle$. The derivation for the subprocess $\mathtt{tick}.\overline{b}\langle\rangle$ should be clear. By assigning the usage to $\mathtt{In}_{[\infty,\infty]}^{[0,0]}$, the cost for $a().\mathtt{tick}.\overline{b}\langle\rangle$ is calculated by: $[\infty,\infty]; K = [0,0]$. Thus, we can correctly infer that the complexity of the deadlocked process is 0. ⌟

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot;\cdot;a:\mathtt{ch}()/0, b:\mathtt{ch}()/\mathtt{Out}_0^{[0,0]} \vdash \overline{b}\langle\rangle \triangleleft [0,0]}}{\cdot;\cdot;a:\mathtt{ch}()/0, b:\mathtt{ch}()/\mathtt{Out}_0^{[1,1]} \vdash \mathtt{tick}.\overline{b}\langle\rangle \triangleleft [1,1]}}{\cdot;\cdot;a:\mathtt{ch}()/\mathtt{In}_{[\infty,\infty]}^{[0,0]}, b:\mathtt{ch}()/\mathtt{Out}_0^{[\infty,\infty]} \vdash a().\mathtt{tick}.\overline{b}\langle\rangle \triangleleft [0,0]} \qquad \text{symmetry for the other branch}}{\dfrac{\cdot;\cdot;a:\mathtt{ch}()/(\mathtt{In}_{[\infty,\infty]}^{[0,0]} \mid \mathtt{Out}_0^{[\infty,\infty]}), b:\mathtt{ch}()/(\mathtt{Out}_0^{[\infty,\infty]} \mid \mathtt{In}_{[0,0]}^{[0,0]}) \vdash a().\mathtt{tick}.\overline{b}\langle\rangle \mid b().\mathtt{tick}.\overline{a}\langle\rangle \triangleleft [0,0]}{\cdot;\cdot;\cdot \vdash (\nu a)(\nu b)(a().\mathtt{tick}.\overline{b}\langle\rangle \mid b().\mathtt{tick}.\overline{a}\langle\rangle) \triangleleft [0,0]}}$$

Figure 9 Typing of Example 22.

▶ **Example 23.** We describe informally an example for which our system can give a complexity, but fails to catch a precise bound. Let us consider the process:

$$P := \mathtt{tick}.!a(n).\mathtt{match}\ n\ \{\mathtt{case}\ 0 \mapsto 0; \mathtt{case}\ \mathtt{s}(m) \mapsto \overline{a}\langle m\rangle\} \mid \overline{a}\langle 10\rangle \mid \mathtt{tick}.\mathtt{tick}.!a(n).0$$

This process has complexity 2. However, if we want to give a usage to the server a, we must have a usage:

$$!\mathtt{In}_0^{[1,1]}.\mathtt{Out}_1^{[0,0]} \mid \mathtt{Out}_{[1,2]}^{[0,0]} \mid !\mathtt{In}_0^{[2,2]}$$

We took as obligations the number of ticks before the action, and as capacity the minimal number for which we have reliability. So in particular, because of the capacity 1 in the usage $\mathtt{Out}_1^{[0,0]}$, typing the recursive call $\overline{a}\langle m\rangle$ increases the complexity by one, and so typing n recursive calls generates a complexity of n in the type system. So, in our setting, the complexity of this process can only be bounded by 10. Overall, this type system may not behave well when there are more than one replicated input process on each server channel, since an imprecision on a capacity for a recursive call leads to an overall imprecision depending on the number of recursive calls. This issue is the only source of imprecision we found with respect to the type system of [4]: see the conjecture in Section 5. ⌟

5 Related Work

Some contributions to the complexity analysis of parallel functional programs by types appear in [16, 20] but the languages studied do not express concurrency. Alternatively [1, 2, 15] address the problem of analysing the time complexity of distributed or concurrent systems. They provide interesting analyses on some instances of systems but do not handle dynamic creation of processes and channel name passing as in the π-calculus. Moreover, the flow graph or rely-guarantee reasoning techniques employed in [1, 2] do not seem to offer the same compositionality as type systems.

There have recently been several studies on type-based cost analysis for binary or multiparty session calculi [5, 9, 10]. It is not clear whether and how those methods can be extended to deal with more general concurrent processes that can be written in the π-calculus, where there may be more than one sender/receiver process for each channel. Among those studies, Das et al.'s work [9] seems technically closest to ours. Both their cost models and ours are parametric, as they rely on a similar `tick` operation. Moreover, their temporal operators seem to have a strong correspondence with usages and operations on them. More specifically, the next operator $\bigcirc$ [9] is similar to the usage operator $\uparrow^{[1,1]}$ in our type system, and $\Box$ and $\Diamond$ roughly correspond to input and output usages with capacity and obligations $[0, \infty]$. For more precise comparison, we need to extend our type system with variant and recursive types, to encode their session calculus into the π-calculus, following [8, 24]. It is left for future work.

Kobayashi et al. [24–26] also used the notion of usages to reason about deadlocks, livelocks, and information flow, but he used a single number for each obligation and capacity (the latter is called a "capability" in his work). In particular, a usage type system for time boundedness, related with parallel complexity, was given in [23]. However, the definition of parallel complexity and thus the definition of usages and reliability in this work is quite different from ours, as its reduction does not take into account some non-deterministic paths. Moreover, as explained in Example 20, the use of a single number and not intervals induces a loss of precision even on simple examples; we have generalized the number to an interval to improve the precision of our analysis. More recently, the first two authors proposed in [4] a type system with the same goal of analysing parallel complexity in π-calculus. This type system builds on sized types and input/output types instead of usages. Because of that, they cannot manage successive uses of the same channel as in Example 2, as their names can essentially be used at only one specific time. In most cases, the time annotation used for channels in their setting corresponds to the sum of the lower bound for obligation and the upper bound for capacity in our setting. We conjecture the following result:

▶ **Conjecture 24** (Comparison with [4]). *Suppose given a typing $\varphi; \Phi; \Gamma_{i/o} \vdash_{i/o} P \triangleleft J$ in the input/output sized type system of [4], such that this process P has a linear use of channels. Then, there exists a reliable context Γ such that $\varphi; \Phi; \Gamma \vdash P \triangleleft [0, J]$.*

More details and some intuitions are given in the Appendix A. So, on a simple use of names our system is strictly more precise if this conjecture is true. However, on other cases, like in Example 23, their system is more precise as the loss of precision because of usage does not happen in their setting. On the contrary, our setting has fairly more precision for processes with a non-trivial use of channels, as in Example 2.

There have also been studies on implicit computational complexity for process calculi, albeit for less expressive calculi than the pi-calculus [7, 14, 28]. Unlike our work, they consider the work rather than the span, and characterize complexity classes, rather than estimating the precise execution time of a given process. The paper [12] by contrast considers the π-calculus and causal (parallel) complexity, but the goal here is also to delineate a characterization of polynomial complexity.

6 Conclusion

We presented a type system built on sized types and usages such that a type derivation for a process gives an upper bound on the parallel complexity of this process. The type system relies on intervals in order to give an approximation of the sizes of integers in the process, and an approximation of the time an input or an output needs to synchronize. In comparison to [4], we showed with examples that our type system can type some concurrent behaviour that was not captured in their type system, and on a certain subset of processes, we conjecture that our new type system is strictly more precise.

Building on previous work by the third author on type inference for usages [25, 27], we plan to investigate type inference, with the use of constraint solving procedures for indices.

References

1 Elvira Albert, Jesús Correas, Einar Broch Johnsen, and Guillermo Román-Díez. Parallel cost analysis of distributed systems. In *Static Analysis - 22nd International Symposium, SAS 2015, Saint-Malo, France, September 9-11, 2015, Proceedings*, volume 9291 of *Lecture Notes in Computer Science*, pages 275–292. Springer, 2015.

2 Elvira Albert, Antonio Flores-Montoya, Samir Genaim, and Enrique Martin-Martin. Rely-guarantee termination and cost analyses of loops with concurrent interleavings. *Journal of Automated Reasoning*, 59(1):47–85, 2017.

3 Martin Avanzini and Ugo Dal Lago. Automating sized-type inference for complexity analysis. *Proceedings of the ACM on Programming Languages*, 1(ICFP):43, 2017.

4 Patrick Baillot and Alexis Ghyselen. Types for complexity of parallel computation in pi-calculus. In *Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021*, volume 12648 of *Lecture Notes in Computer Science*, pages 59–86. Springer, 2021.

5 David Castro-Perez and Nobuko Yoshida. CAMP: cost-aware multiparty session protocols. *Proc. ACM Program. Lang.*, 4(OOPSLA):155:1–155:30, 2020.

6 Ugo Dal Lago and Marco Gaboardi. Linear dependent types and relative completeness. In *Logic in Computer Science (LICS), 2011 26th Annual IEEE Symposium on*, pages 133–142. IEEE, 2011.

7 Ugo Dal Lago, Simone Martini, and Davide Sangiorgi. Light logics and higher-order processes. *Mathematical Structures in Computer Science*, 26(6):969–992, 2016.

8 Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. *Information and Computation*, 256:253–286, 2017.

9 Ankush Das, Jan Hoffmann, and Frank Pfenning. Parallel complexity analysis with temporal session types. *Proc. ACM Program. Lang.*, 2(ICFP):91:1–91:30, 2018.

10 Ankush Das, Jan Hoffmann, and Frank Pfenning. Work analysis with resource-aware session types. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 305–314. ACM, 2018.

11 Romain Demangeon, Daniel Hirschkoff, Naoki Kobayashi, and Davide Sangiorgi. On the complexity of termination inference for processes. In Gilles Barthe and Cédric Fournet, editors, *Trustworthy Global Computing, Third Symposium, TGC 2007, Sophia-Antipolis, France, November 5-6, 2007, Revised Selected Papers*, volume 4912 of *Lecture Notes in Computer Science*, pages 140–155. Springer, 2007. doi:10.1007/978-3-540-78663-4_11.

12 Romain Demangeon and Nobuko Yoshida. Causal computational complexity of distributed processes. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 344–353. ACM, 2018.

13 Yuxin Deng and Davide Sangiorgi. Ensuring termination by typability. *Information and Computation*, 204(7):1045–1082, 2006.

14 Paolo Di Giamberardino and Ugo Dal Lago. On session types and polynomial time. *Mathematical Structures in Computer Science*, -1, 2015.

15 Elena Giachino, Einar Broch Johnsen, Cosimo Laneve, and Ka I Pun. Time complexity of concurrent programs - - A technique based on behavioural types -. In *Formal Aspects of Component Software - 12th International Conference, FACS 2015, Niterói, Brazil, October 14-16, 2015, Revised Selected Papers*, volume 9539 of *Lecture Notes in Computer Science*, pages 199–216. Springer, 2016.

16 Stéphane Gimenez and Georg Moser. The complexity of interaction. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 243–255, 2016.

17 Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Multivariate amortized resource analysis. *ACM Trans. Program. Lang. Syst.*, 34(3):14:1–14:62, 2012.

18 Jan Hoffmann, Klaus Aehlig, and Martin Hofmann. Resource aware ML. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 781–786. Springer, 2012.

19 Jan Hoffmann and Martin Hofmann. Amortized resource analysis with polynomial potential. In *Programming Languages and Systems, 19th European Symposium on Programming, ESOP 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings*, volume 6012 of *Lecture Notes in Computer Science*, pages 287–306. Springer, 2010.

20 Jan Hoffmann and Zhong Shao. Automatic static cost analysis for parallel programs. In Jan Vitek, editor, *Programming Languages and Systems*, pages 132–157, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

21 Martin Hofmann and Steffen Jost. Static prediction of heap space usage for first-order functional programs. In *Conference Record of POPL 2003: The 30th SIGPLAN-SIGACT Symposium on Principles of Programming Languages, New Orleans, Louisisana, USA, January 15-17, 2003*, pages 185–197. ACM, 2003.

22 Naoki Kobayashi. A partially deadlock-free typed process calculus. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*, pages 128–139. IEEE Computer Society, 1997. doi:10.1109/LICS.1997.614941.

23 Naoki Kobayashi. A type system for lock-free processes. *Information and Computation*, 177(2):122–159, 2002.

24 Naoki Kobayashi. Type systems for concurrent programs. In *Formal Methods at the Crossroads. From Panacea to Foundational Support*, pages 439–453. Springer, 2003.

25 Naoki Kobayashi. Type-based information flow analysis for the π-calculus. *Acta Informatica*, 42(4-5):291–347, 2005.

26 Naoki Kobayashi. A new type system for deadlock-free processes. In *International Conference on Concurrency Theory*, pages 233–247. Springer, 2006.

27 Naoki Kobayashi, Shin Saito, and Eijiro Sumii. An implicitly-typed deadlock-free process calculus. In Catuscia Palamidessi, editor, *CONCUR 2000 — Concurrency Theory*, pages 489–504. Springer Berlin Heidelberg, 2000.

28 Antoine Madet and Roberto M. Amadio. An elementary affine λ-calculus with multithreading and side effects. In *Typed Lambda Calculi and Applications - 10th International Conference, TLCA 2011, Novi Sad, Serbia, June 1-3, 2011. Proceedings*, volume 6690 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2011.

29 Robin Milner. *Communication and concurrency.* PHI Series in computer science. Prentice Hall, 1989.

30 Eijiro Sumii and Naoki Kobayashi. A generalized deadlock-free process calculus. In *Proc. of Workshop on High-Level Concurrent Language (HLCL'98)*, volume 16(3) of *ENTCS*, pages 55–77, 1998.

A Comparison with [4]

In this section, we give intuitively a description of how to simulate types on [4] in a linear setting with usage. We say that a process has a linear use of channels if it use channel names at most one time for input and at most one time for output. For servers, we suppose that the replicated input is once and for all defined at the beginning of a process, and as free variables it can only use others servers. In their type system, a channel is given a type $\mathtt{Ch}_I(\widetilde{T})$ where I is an upper bound on the time this channel communicates. It can also be a variant of this type with only input or only output capability. Such a channel would be represented in out type system by a type $\mathtt{ch}(\underline{\widetilde{T}})/(\mathtt{In}^{[I_1,I_1]}_{J^1_c} \mid \mathtt{Out}^{I_2,I_2}_{J^2_c})$ where either J^1_c is 0 and then $I_1 \leq I$, either $J^1_c = [J_1, J_1]$ and then $I_1 + J_1 \leq I$. We have the same thing for J^2_c and I_2. To be more precise, the typing in our setting should be a non-deterministic choice (using $+$) over such usages, and the capacity should adapt to the obligation of the dual action in order to be reliable. So, for example if $I_1 \leq I_2$, then we would take: $\mathtt{ch}(\underline{\widetilde{T}})/(\mathtt{In}^{[I_1,I_1]}_{[I_2-I_1,I_2-I_1]} \mid \mathtt{Out}^{I_2,I_2}_{0})$. Note that this shape of type adapts well to the way time is delayed in their setting. For example, the `tick` constructor in their setting make the time advance by 1, and in our setting, then we would obtain the usage $(\mathtt{In}^{[I_1+1,I_1+1]}_{[I_2-I_1,I_2-I_1]} \mid \mathtt{Out}^{I_2+1,I_2+1}_{0})$ and we still have $I_2 - I_1 = (I_2+1) - (I_1+1)$.

In the same way, in their setting when doing an output (or input), the time is delayed by I. Here, with usages, it would be delayed by J_c which is, by definition, a delay of the shape $\uparrow^{[J,J]}$ with $J \leq I$. So, we would keep the invariant that our time annotation have the shape of singleton interval with a smaller value than the time annotation in their setting.

For servers, in the linear setting, their types have the shape: ${}_I\forall\widetilde{i}.\mathtt{srv}^J(\widetilde{T})$ where $I = 0$ is again a time annotation giving an upper bound on the time the input action of this server is defined, and J is a complexity as in our setting. So, in our setting it would be:

$$\forall\widetilde{i}.\mathtt{srv}^{[0,J]}(\widetilde{\underline{T}})/!\mathtt{In}_{\infty}^{[0,0]}.!\mathtt{Out}_{0}^{[0,\infty]} \mid !\mathtt{Out}_{0}^{[0,\infty]}$$

Note that this usage is reliable. The main point here is this infinite capacity for input. Please note that because of our input rule for servers, it does not generates an infinite complexity. However, it imposes a delaying $\uparrow^{[0,\infty]}!\Gamma$ in the context. Because of the shape we gave to types, it means that the context can only have outputs for other servers as free variables, but this was the condition imposed by linearity. Note that in [4], they have a restriction on the free variables of servers that is in fact the same restriction so it does not harm the comparison to take this restriction on free variables. As an example, the bitonic sort described in [4] could be typed similarly in our setting with this kind of type.

Finally, choice in usages $U_1 + U_2$ is used to put together the different usages we obtain in the two branches of a pattern matching.

B Proofs

In this section, we prove Theorem 18, after giving various lemmas.

B.1 Properties of Subusage

The subusage relation satisfies some properties that are essential for the soundness theorem. First, we have the usual properties of subusage:

▶ **Lemma 25.** *If $\varphi; \Phi \vDash B_o \subseteq A_o$ then $\varphi; \Phi \vdash (\uparrow^{A_o} U) \sqsubseteq (\uparrow^{B_o} U)$*

▶ **Lemma 26** (Properties of Subusage)**.** *For a set of index variables φ and a set of constraints Φ on φ we have:*

1. *If $\varphi; \Phi \vdash U \sqsubseteq V$ then for any interval A_o, we have $\varphi; \Phi \vdash \uparrow^{A_o} U \sqsubseteq \uparrow^{A_o} V$.*
2. *If $\varphi; \Phi \vdash U \sqsubseteq V$ and $\varphi; \Phi \vdash V \longrightarrow V'$, then there exists U' such that $\varphi; \Phi \vdash U \longrightarrow^* U'$ and $\varphi; \Phi \vdash U' \sqsubseteq V'$ (with $\mathtt{err} \sqsubseteq U$ for any usage U)*
3. *If $\varphi; \Phi \vdash U \sqsubseteq V$ and U is reliable under $\varphi; \Phi$ then V is reliable under $\varphi; \Phi$.*

The proof of the first lemma is rather easy, but for the second lemma it is a bit cumbersome. Intuitively all the base rules for subusage satisfy this and the main difficulty is to show that the congruence rule and context rule conserve those properties.

More specific to this type system, we also have the following lemma, stating that delaying does not modify the behaviour of a type.

▶ **Lemma 27** (Invariance by Delaying). *For any interval A_o:*

1. *If $\varphi; \Phi \vdash \uparrow^{A_o} U \longrightarrow V'$ then, there exists V such that $\uparrow^{A_o} V = V'$ and $\varphi; \Phi \vdash U \longrightarrow V$. (with $\mathtt{err} = \uparrow^{A_o} \mathtt{err}$)*
2. *If $\varphi; \Phi \vdash U \longrightarrow V$ then $\varphi; \Phi \vdash \uparrow^{A_o} U \longrightarrow \uparrow^{A_o} V$.*
3. *U is reliable under $\varphi; \Phi$ if and only if $\uparrow^{A_o} U$ is reliable under $\varphi; \Phi$.*

In our setting, this lemma shows among other things that the `tick` constructor, or more generally the annotation $n : P$, does not break reliability. Those properties can easily be proved with simple inductions.

B.2 Intermediate Lemmas

We give some intermediate lemmas for the soundness theorem

▶ **Lemma 28** (Weakening). *Let φ, φ' be disjoint set of index variables, Φ be a set of constraints on φ, Φ' be a set of constraints on (φ, φ'), Γ and Γ' be contexts on disjoint set of variables.*

1. *If $\varphi; \Phi; \Gamma \vdash e : T$ then $(\varphi, \varphi'); (\Phi, \Phi'); \Gamma, \Gamma' \vdash e : T$.*
2. *If $\varphi; \Phi; \Gamma \vdash P \triangleleft K$ then $(\varphi, \varphi'); (\Phi, \Phi'); \Gamma, \Gamma' \vdash P \triangleleft K$.*

We also show that we can remove some useless hypothesis.

▶ **Lemma 29** (Strengthening). *Let φ be a set of index variables, Φ be a set of constraints on φ, and C a constraint on φ such that $\varphi; \Phi \vDash C$.*

1. *If $\varphi; (\Phi, C); \Gamma, \Gamma' \vdash e : T$ and the variables in Γ' are not free in e, then $\varphi; \Phi; \Gamma \vdash e : T$.*
2. *If $\varphi; (\Phi, C); \Gamma, \Gamma' \vdash P \triangleleft K$ and the variables in Γ' are not free in P, then $\varphi; \Phi; \Gamma \vdash P \triangleleft K$.*

Those two lemmas are proved easily by successive induction on the definitions in this paper. Then, we also have a lemma expressing that index variables can indeed be replaced by any index.

▶ **Lemma 30** (Index Substitution). *Let φ be a set of index variables and $i \notin \varphi$. Let $J_{\mathbb{N}}$ be an index with free variables in φ. Then,*

1. *If $(\varphi, i); \Phi; \Gamma \vdash e : T$ then $\varphi; \Phi\{J_{\mathbb{N}}/i\}; \Gamma\{J_{\mathbb{N}}/i\} \vdash e : T\{J_{\mathbb{N}}/i\}$.*
2. *If $(\varphi, i); \Phi; \Gamma \vdash P \triangleleft K$ then $\varphi; \Phi\{J_{\mathbb{N}}/i\}; \Gamma\{J_{\mathbb{N}}/i\} \vdash P \triangleleft K\{J_{\mathbb{N}}/i\}$.*

B.3 Substitution Lemma

We now present the variable substitution lemmas. In the setting of usages, this lemma is a bit more complex than usual. Indeed, we have a separation of contexts with the parallel composition, and we have to rely on subusage, especially the rule $\varphi; \Phi \vdash (\alpha_J^{A_o}.U) \mid (\uparrow^{A_o + J_c} V) \sqsubseteq \alpha_{J_c}^{A_o}.(U \mid V)$ as expressed in the Example 11 above. We put some emphasis on the following notation: when we write $\Gamma, v : T$ as a context in typing, it means that v does not appear in Γ.

▶ **Lemma 31** (Substitution). *Let Γ and Δ be contexts such that $\Gamma \mid \Delta$ is defined. Then we have:*

1. *If $\varphi; \Phi; \Gamma, v : T \vdash e' : T'$ and $\Delta \vdash e : T$ then $\varphi; \Phi; \Gamma \mid \Delta \vdash e'[v := e] : T'$*
2. *If $\varphi; \Phi; \Gamma, v : T \vdash P \triangleleft K$ and $\Delta \vdash e : T$ then $\varphi; \Phi; \Gamma \mid \Delta \vdash P[v := e] \triangleleft K$*

The first point is straightforward. It uses the fact that we have the relation $\varphi; \Phi \vdash U \sqsubseteq 0$ for any usage U, and so we can use $\varphi; \Phi \vdash \Gamma \mid \Delta \sqsubseteq \Gamma$ in order to weaken Δ (similarly for Γ) if needed. The second point is more interesting. The easy case is when T is $\mathsf{Nat}[I, J]$ for some $[I, J]$. Then, we take a Δ that only uses the zero usage, and so $\Gamma \mid \Delta = \Gamma$ and everything becomes simpler. The more interesting cases are:

▶ **Lemma 32** (Difficult Cases of Substitution). *We have:*

- *If* $\varphi; \Phi; \Gamma, b{:}\, \boldsymbol{ch}(\widetilde{S})/W_0, c{:}\, \boldsymbol{ch}(\widetilde{S})/W_1 \vdash P \triangleleft K$ *then* $\varphi; \Phi; \Gamma, b{:}\, \boldsymbol{ch}(\widetilde{S})/(W_0 \mid W_1) \vdash P[c := b] \triangleleft K$
- *If* $\varphi; \Phi; \Gamma, b : \forall \widetilde{i}.\boldsymbol{srv}^K(\widetilde{S})/W_0, c : \forall \widetilde{i}.\boldsymbol{srv}^K(\widetilde{S})/W_1 \vdash P \triangleleft K$ *then* $\varphi; \Phi; \Gamma, b : \forall \widetilde{i}.\boldsymbol{srv}^K(\widetilde{S})/(W_0 \mid W_1) \vdash P[c := b] \triangleleft K$

With a careful induction, we can indeed prove this lemma, relying on the subusage relation.

B.4 Congruence Equivalence

In order to prove the soundness theorem, we need first a lemma saying that the congruence relation behaves well with typing.

▶ **Lemma 33** (Congruence and Typing). *Let* P *and* Q *be annotated processes such that* $P \equiv Q$. *Then,* $\varphi; \Phi; \Gamma \vdash P \triangleleft K$ *if and only if* $\varphi; \Phi; \Gamma \vdash Q \triangleleft K$.

The proof is an induction on $P \equiv Q$, relying on all the previous lemma, and especially the lemmas on delaying for congruence rules specific to annotated processes.

B.5 Proof of Theorem 18 (Subject Reduction)

We now detail some cases for the proof of Theorem 18.

Note that for a process P, the typing system is not syntax-directed because of the subtyping rule. However, by reflexivity and transitivity of subtyping, we can always assume that a proof has exactly *one* subtyping rule before any syntax-directed rule. Moreover, notice that in those kinds of proof, the top-level rule of subtyping can be ignored. Indeed, we can always simulate exactly the same subtyping rule for both P and Q We now proceed by doing the case analysis on the rules of Figure 1. In order to simplify the proof, we will also consider that types and indexes invariant by subtyping (like the complexity in a server) are not renamed with subtyping. Note that this only add cumbersome notations but it does not change the core of the proof. We detail only the case for synchronization:

Case $(n : a(\widetilde{v}).P) \mid (m : \overline{a}\langle\widetilde{e}\rangle.Q) \Rightarrow (\max(m,n) : (P[\widetilde{v} := \widetilde{e}] \mid Q))$. Consider the typing $\varphi; \Phi; \Gamma_0 \mid \Delta_0, a : \mathtt{ch}(\widetilde{T})/(U_0 \mid V_0) \vdash (n : a(\widetilde{v}).P) \mid (m : \overline{a}\langle\widetilde{e}\rangle.Q) \triangleleft K_0 \sqcup K_0'$. The first rule is the rule for parallel composition, then the proof is split into the two following subtree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi_P}{\varphi; \Phi; \Gamma_2, a : \mathtt{ch}(\widetilde{T})/U_2, \widetilde{v} : \widetilde{T} \vdash P \triangleleft K_2}}{\varphi; \Phi; \uparrow^{J_c}\Gamma_2, a : \mathtt{ch}(\widetilde{T})/\mathtt{In}^{[0,0]}_{J_c}.U_2 \vdash a(\widetilde{v}).P \triangleleft J_c; K_2 \qquad (3)}}{\varphi; \Phi; \Gamma_1, a : \mathtt{ch}(\widetilde{T})/U_1 \vdash a(\widetilde{v}).P \triangleleft K_1}}{\dfrac{\varphi; \Phi; \uparrow^{[n,n]}\Gamma_1, a : \mathtt{ch}(\widetilde{T})/\uparrow^{[n,n]}U_1 \vdash n : a(\widetilde{v}).P \triangleleft K_1 + [n,n] \qquad (4)}{\varphi; \Phi; \Gamma_0, a : \mathtt{ch}(\widetilde{T})/U_0 \vdash n : a(\widetilde{v}).P \triangleleft K_0}}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi_e}{\varphi; \Phi; \Delta_2, a : \mathtt{ch}(\widetilde{T})/V_2 \vdash \widetilde{e} : \widetilde{T}} \qquad \dfrac{\pi_Q}{\varphi; \Phi; \Delta_2', a : \mathtt{ch}(\widetilde{T})/V_2' \vdash Q \triangleleft K_2'}}{\varphi; \Phi; \uparrow^{J_c'}(\Delta_2 \mid \Delta_2'), a : \mathtt{ch}(\widetilde{T})/\mathtt{Out}^{[0,0]}_{J_c'}.(V_2 \mid V_2') \vdash \overline{a}\langle\widetilde{e}\rangle.Q \triangleleft J_c'; K_2' \qquad (1)}}{\varphi; \Phi; \Delta_1, a : \mathtt{ch}(\widetilde{T})/V_1 \vdash \overline{a}\langle\widetilde{e}\rangle.Q \triangleleft K_1'}}{\dfrac{\varphi; \Phi; \uparrow^{[m,m]}\Delta_1, a : \mathtt{ch}(\widetilde{T})/\uparrow^{[m,m]}V_1 \vdash m : \overline{a}\langle\widetilde{e}\rangle.Q \triangleleft K_1' + [m,m] \qquad (2)}{\varphi; \Phi; \Delta_0, a : \mathtt{ch}(\widetilde{T})/V_0 \vdash m : \overline{a}\langle\widetilde{e}\rangle.Q \triangleleft K_0'}}$$

where (1) is:

$$\varphi; \Phi \vdash \Delta_1 \sqsubseteq \uparrow^{J'_c}(\Delta_2 \mid \Delta'_2) \qquad \varphi; \Phi \vdash V_1 \sqsubseteq \mathtt{Out}^{[0,0]}_{J'_c}.(V_2 \mid V'_2) \qquad \varphi; \Phi \vDash J'_c; K'_2 \subseteq K'_1$$

(2) is:

$$\varphi; \Phi \vdash \Delta_0 \sqsubseteq \uparrow^{[n,n]}\Delta_1; V_0 \sqsubseteq \uparrow^{[m,m]}V_1; K'_1 + [m,m] \subseteq K'_0$$

(3) is:

$$\varphi; \Phi \vdash \Gamma_1 \sqsubseteq \uparrow^{J_c}\Gamma_2; U_1 \sqsubseteq \mathtt{In}^{[0,0]}_{J_c}.U_2; J_c; K_2 \subseteq K_1$$

(4) is:

$$\varphi; \Phi \vdash \Gamma_0 \sqsubseteq \uparrow^{[n,n]}\Gamma_1; U_0 \sqsubseteq \uparrow^{[n,n]}U_1; K_1 + [n,n] \subseteq K_0$$

First, we know that $\Gamma_0 \mid \Delta_0$ is defined. Moreover, we have

$$\varphi; \Phi \vdash \Gamma_0 \sqsubseteq \uparrow^{[n,n]}\Gamma_1 \qquad \varphi; \Phi \vdash \Gamma_1 \sqsubseteq \uparrow^{J_c}\Gamma_2 \qquad \varphi; \Phi \vdash \Delta_0 \sqsubseteq \uparrow^{[m,m]}\Delta_1 \qquad \Delta_1 \sqsubseteq \uparrow^{J'_c}(\Delta_2 \mid \Delta'_2)$$

So, for the channel and server types, in those seven contexts, the shape of the type does not change (only the usage can change). We also have:

$$\Gamma_0^{\mathsf{Nat}} = \Delta_0^{\mathsf{Nat}} \qquad \varphi; \Phi \vdash \Gamma_0^{\mathsf{Nat}} \sqsubseteq \Gamma_1^{\mathsf{Nat}} \sqsubseteq \Gamma_2^{\mathsf{Nat}} \qquad \varphi; \Phi \vdash \Delta_0^{\mathsf{Nat}} \sqsubseteq \Delta_1^{\mathsf{Nat}} \sqsubseteq \Delta_2^{\mathsf{Nat}} \qquad \Delta_2^{\mathsf{Nat}} = \Delta_2'^{\mathsf{Nat}}$$

So, from π_e and π_P we obtain by subtyping:

$$\varphi; \Phi; \Gamma_0^{\mathsf{Nat}}, \Gamma_2^{\nu}, a : \mathtt{ch}(\widetilde{T})/U_2, \widetilde{v} : \widetilde{T} \vdash P \triangleleft K_2 \qquad \varphi; \Phi; \Gamma_0^{\mathsf{Nat}}, \Delta_2^{\nu}, a : \mathtt{ch}(\widetilde{T})/V_2 \vdash \widetilde{e} : \widetilde{T}$$

So, we use the substitution lemma (Lemma 31) and we obtain:

$$\varphi; \Phi; \Gamma_0^{\mathsf{Nat}}, (\Gamma_2^{\nu} \mid \Delta_2^{\nu}), a : \mathtt{ch}(\widetilde{T})/(U_2 \mid V_2) \vdash P[\widetilde{v} := \widetilde{e}] \triangleleft K_2$$

As previously, by subtyping from π_Q, we have:

$$\varphi; \Phi; \Gamma_0^{\mathsf{Nat}}, \Delta_2'^{\nu}, a : \mathtt{ch}(\widetilde{T})/V'_2 \vdash Q \triangleleft K'_2$$

Thus, with the parallel composition rule (as parallel composition of context is defined) and subtyping we have:

$$\varphi; \Phi; \Gamma_0^{\mathsf{Nat}}, (\Gamma_2^{\nu} \mid \Delta_2^{\nu} \mid \Delta_2'^{\nu}), a : \mathtt{ch}(\widetilde{T})/(U_2 \mid V_2 \mid V'_2) \vdash (P[\widetilde{v} := \widetilde{e}] \mid Q) \triangleleft K_2 \sqcup K'_2$$

Let us denote $M = \max(m, n)$. Thus, we derive the proof:

$$\frac{\varphi; \Phi; \Gamma_0^{\mathsf{Nat}}, (\Gamma_2^{\nu} \mid \Delta_2^{\nu} \mid \Delta_2'^{\nu}), a : \mathtt{ch}(\widetilde{T})/(U_2 | V_2 V'_2) \vdash (P[\widetilde{v} := \widetilde{e}] \mid Q) \triangleleft K_2 \sqcup K'_2}{\varphi; \Phi; \Gamma_0^{\mathsf{Nat}}, \uparrow^{[M,M]}(\Gamma_2^{\nu} | \Delta_2^{\nu} | \Delta_2'^{\nu}), a : \mathtt{ch}(\widetilde{T})/\uparrow^{[M,M]}(U_2 | V_2 | V'_2) \vdash M : (P[\widetilde{v} := \widetilde{e}] \mid Q) \triangleleft (K_2 \sqcup K'_2) + [M,M]}$$

Now, recall that by hypothesis, $U_0 \mid V_0$ is reliable. We have:

$$\varphi; \Phi \vdash U_0 \sqsubseteq \uparrow^{[n,n]}U_1 \qquad \varphi; \Phi \vdash U_1 \sqsubseteq \mathtt{In}^{[0,0]}_{J_c}.U_2 \qquad \varphi; \Phi \vdash V_0 \sqsubseteq \uparrow^{[m,m]}V_1 \qquad \varphi; \Phi \vdash V_1 \sqsubseteq \mathtt{Out}^{[0,0]}_{J'_c}(V_2 \mid V'_2)$$

So, by Point 1 of Lemma 26, with transitivity and parallel composition of subusage, we have:

$$\varphi; \Phi \vdash U_0 \mid V_0 \sqsubseteq (\uparrow^{[n,n]}U_1) \mid (\uparrow^{[m,m]}V_1) \sqsubseteq \mathtt{In}^{[n,n]}_{J_c}.U_2 \mid \mathtt{Out}^{[m,m]}_{J'_c}(V_2 \mid V'_2)$$

By Point 3 of Lemma 26, we have $\mathtt{In}^{[n,n]}_{J_c}.U_2 \mid \mathtt{Out}^{[m,m]}_{J'_c}(V_2 \mid V'_2)$ reliable. So, in particular, we have:

$$\varphi; \Phi \vdash \mathtt{In}^{[n,n]}_{J_c}.U_2 \mid \mathtt{Out}^{[m,m]}_{J'_c}(V_2 \mid V'_2) \longrightarrow \uparrow^{[M,M]}(U_2 \mid V_2 \mid V'_2)$$

$$\varphi; \Phi \vDash [n,n] \subseteq [m,m] \oplus J'_c \qquad \varphi; \Phi \vDash [m,m] \subseteq [n,n] \oplus J_c$$

Thus, we deduce that

$$\varphi; \Phi \vDash [M,M] \subseteq [n,n] + J_c \qquad \varphi; \Phi \vDash [M,M] \subseteq [m,m] + J'_c$$

So, we have in particular, with Lemma 25 and Point 1 of Lemma 26 and parallel composition:

$$\varphi; \Phi \vdash \Gamma_0 \mid \Delta_0 \sqsubseteq (\uparrow^{[n,n]}\Gamma_1) \mid \uparrow^{[m,m]}\Delta_1 \sqsubseteq (\uparrow^{[n,n]+J_c}\Gamma_2) \mid (\uparrow^{[m,m]+J'_c}(\Delta_2 \mid \Delta'_2)) \sqsubseteq \uparrow^{[M,M]}(\Gamma_2 \mid \Delta_2 \mid \Delta'_2)$$

We also have

$$\varphi; \Phi \vDash K_2 + [M,M] \subseteq J_c; K_2 + [n,n] \subseteq K_0 \qquad \varphi; \Phi \vDash K'_2 + [M,M] \subseteq J'_c; K'_2 + [m,m] \subseteq K'_0$$

So, we obtain directly $\varphi; \Phi \vDash (K_2 \sqcup K'_2) + [M,M] \subseteq K_0 \sqcup K'_0$

Thus, we can simplify a bit the derivation given above, and we have:

$$\dfrac{\dfrac{\varphi; \Phi; \Gamma_0^{\mathsf{Nat}}, (\Gamma_2^{\nu} \mid \Delta_2^{\nu} \mid \Delta_2'^{\nu}), a : \mathtt{ch}(\widetilde{T})/(U_2 \mid V_2 \mid V'_2) \vdash (P[\widetilde{v} := \widetilde{e}] \mid Q) \triangleleft K_2 \sqcup K'_2}{\varphi; \Phi; \Gamma_0^{\mathsf{Nat}}, \uparrow^{[M,M]}(\Gamma_2^{\nu} | \Delta_2^{\nu} | \Delta_2'^{\nu}), a : \mathtt{ch}(\widetilde{T})/\uparrow^{[M,M]}(U_2 | V_2 | V'_2) \vdash M : (P[\widetilde{v} := \widetilde{e}] \mid Q) \triangleleft (K_2 \sqcup K'_2) + [M,M]}}{\varphi; \Phi; (\Gamma_0 \mid \Delta_0), a : \mathtt{ch}(\widetilde{T})/\uparrow^{[M,M]}(U_2 \mid V_2 \mid V'_2) \vdash M : (P[\widetilde{v} := \widetilde{e}] \mid Q) \triangleleft K_0 \sqcup K'_0}$$

By Point 2 of Lemma 26, there exists W such that:

$$\varphi; \Phi \vdash U_0 \mid V_0 \longrightarrow^{*} W \qquad \varphi; \Phi \vdash W \sqsubseteq \uparrow^{[M,M]}(U_2 \mid V_2 \mid V'_2)$$

So, by subtyping we have a proof:

$$\varphi; \Phi; \Gamma_0 \mid \Delta_0, a : \mathtt{ch}(\widetilde{T})/W \vdash M : (P[\widetilde{v} := \widetilde{e}] \mid Q) \triangleleft K_0 \sqcup K'_0$$

This concludes this case.

Generalising Projection in Asynchronous Multiparty Session Types

Rupak Majumdar
Max Planck Institute for Software Systems, Kaiserslautern, Germany

Madhavan Mukund
Chennai Mathematical Institute, India
CNRS IRL 2000, ReLaX, Chennai, India

Felix Stutz
Max Planck Institute for Software Systems, Kaiserslautern, Germany

Damien Zufferey
Max Planck Institute for Software Systems, Kaiserslautern, Germany

Abstract

Multiparty session types (MSTs) provide an efficient methodology for specifying and verifying message passing software systems. In the theory of MSTs, a global type specifies the interaction among the roles at the global level. A local specification for each role is generated by projecting from the global type on to the message exchanges it participates in. Whenever a global type can be projected on to each role, the composition of the projections is deadlock free and has exactly the behaviours specified by the global type. The key to the usability of MSTs is the projection operation: a more expressive projection allows more systems to be type-checked but requires a more difficult soundness argument.

In this paper, we generalise the standard projection operation in MSTs. This allows us to model and type-check many design patterns in distributed systems, such as load balancing, that are rejected by the standard projection. The key to the new projection is an analysis that tracks causality between messages. Our soundness proof uses novel graph-theoretic techniques from the theory of message-sequence charts. We demonstrate the efficacy of the new projection operation by showing many global types for common patterns that can be projected under our projection but not under the standard projection operation.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases Multiparty session types, Verification, Communicating state machines

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.35

Related Version *Extended Version with Proofs*: `https://arxiv.org/abs/2107.03984`

Supplementary Material Tool available at:
Software (Source Code): `https://doi.org/10.5281/zenodo.5144684`

Funding This research was funded in part by the Deutsche Forschungsgemeinschaft project 389792660-TRR 248 and by the European Research Council under ERC Synergy Grant ImPACT (610150).

Acknowledgements The authors would like to thank Nobuko Yoshida, Thomas Wies, Elaine Li, and the anonymous reviewers for their feedback and suggestions.

1 Introduction

Distributed message-passing systems are both widespread and challenging to design and implement. A process tries to implement its role in a protocol with only the partial information received through messages. The unpredictable communication delays mean that messages from different sources can be arbitrarily reordered. Combining concurrency, asynchrony, and message buffering makes the verification problem algorithmically undecidable [10] and principled design and verification of such systems is an important challenge.

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 35; pp. 35:1–35:24
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Multiparty Session Types (MSTs) [37, 52] provide an appealing type-based approach for formalising and compositionally verifying structured concurrent distributed systems. They have been successfully applied to web services [58], distributed algorithms [41], smart contracts [23], operating systems [26], high performance computing [34], timed systems [8], cyber-physical systems [47], etc. By decomposing the problem of asynchronous verification on to local roles, MSTs provide a clean and modular approach to the verification of distributed systems (see the surveys [4, 39]).

The key step in MSTs is the *projection* from a *global type*, specifying all possible global message exchanges, to *local types* for each role. The soundness theorem of MSTs states that every projectable global type is *implementable*: there is a distributed implementation that is free from communication safety errors such as deadlocks and unexpected messages.

The projection keeps only the operations observable by a given role and yet maintains the invariant that every choice can be distinguished in an unambiguous way. Most current projection operations ensure this invariant by syntactically merging different paths locally for each role. While these projections are syntactic and efficient, they are also very conservative and disallow many common design patterns in distributed systems.

In this paper, we describe a more general projection for MSTs to address the conservatism of existing projections. To motivate our extension, consider a simple load balancing protocol: a client sends a request to a server and the server forwards the request to one of two workers. The workers serve the request and directly reply to the client. (We provide the formal syntax later.) This common protocol is disallowed by existing MST systems, either because they syntactically disallow such messages (the *directed choice* restriction that states the sender and recipient must be the same along every branch of a choice), or because the projection operates only on the global type and disallows inferred choice.

The key difficulty in projection is to manage the interaction between choice and concurrency in a distributed setting. Without choice, all roles would just follow one predetermined sequence of send and receive operations. Introducing choice means a role either decides whom to send which message next, or reacts to the choices of other roles – even if such choices are not locally visible. This is only possible when the outcome of every choice propagates unambiguously. At each point, every role either is agnostic to a prior choice or knows exactly the outcome of the choice, even though it may only receive information about the choice indirectly through subsequent communication with other roles. Unfortunately, computing how choice propagates in a system is undecidable in general [3]; this is the reason why conservative restrictions are used in practice.

The key insight in our projection operation is to manage the interaction of choice and concurrency via a *message causality analysis*, inspired by the theory of communicating state machines (CSMs) and message sequence charts (MSCs), that provides a more global view. We resolve choice based on *available messages* along different branches. The causality analysis provides more information when merging two paths based on expected messages.

We show that our generalised projection subsumes previous approaches that lift the directed choice restriction [16, 38, 17, 40]. Empirically, it allows us to model and verify common distributed programming patterns such as load balancing, replicated data, and caching – where a server needs to choose between different workers – that are not in scope of current MSTs, while preserving the efficiency of projection.

We show type soundness for generalised projection. This generalisation is non-trivial, since soundness depends on subtle arguments about asynchronous messages in the system. We prove the result using an automata-theoretic approach, also inspired by the theory of MSCs, that argues about traces in communicating state machines. Our language-theoretic proof is different from the usual proof-theoretic approaches in soundness proofs of MST systems, and builds upon technical machinery from the theory of MSCs.

We show empirically that generalised choice is key to modelling several interesting instances in distributed systems while maintaining the efficiency of more conservative systems. Our global type specifications go beyond existing examples in the literature of MSTs.

2 Multiparty Session Types with Generalised Choice

In this section, we define global and local types. We explain how multiparty session types (MSTs) work and present a shortcoming of current MSTs. Our MSTs overcome this shortcoming by allowing a role to wait for messages coming from different senders. We define a new projection operation from global to local types: the projection represents global message exchanges from the perspective of a single role. The key to the new projection is a generalised *merge* operator that prevents confusion between messages from different senders.

2.1 Global and Local Types

We describe the syntax of global types following work by Honda et al. [36], Hu and Yoshida [38], and Scalas and Yoshida [52]. We focus on the core message-passing aspects of asynchronous MSTs and do not include features such as delegation.

▶ **Definition 1** (Syntax). Global types for MSTs *are defined by the grammar:*

$$G ::= 0 \mid \sum_{i \in I} \mathtt{p} \to \mathtt{q}_i : m_i . G_i \mid \mu t.\, G \mid t$$

where $\mathtt{p}, \mathtt{q}_i$ *range over a set of roles* $\mathcal{P}$*,* m_i *over a set of messages* $\mathcal{V}$*, and* t *over type variables.*

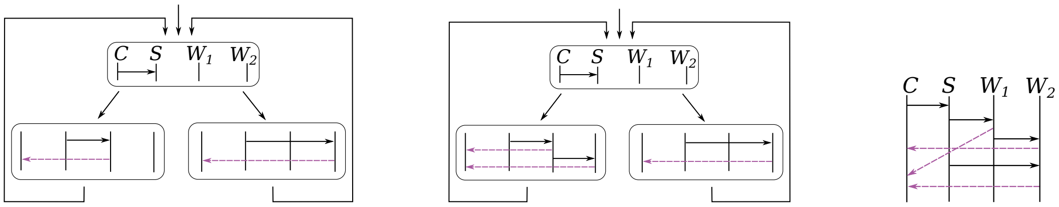
Note that our definition of global types extends the standard syntax (see, e.g., [36]), which has a *directed choice* restriction, requiring that a sender must send messages to the same role along different branches of a choice. Our syntax $\sum_{i \in I} \mathtt{p} \to \mathtt{q}_i : m_i . G_i$ allows a sender to send messages to different roles along different branches (as in, e.g., [38]). For readability, we sometimes use the infix operator $+$ for choice, instead of $\sum$. When $|I| = 1$, we omit $\sum$.

In a global type, the send and the receive operations of a message exchange are specified atomically. An expression $\mathtt{p} \to \mathtt{q} : m$ represents two events: a *send* $\mathtt{p} \triangleright \mathtt{q}!m$ and a *receive* $\mathtt{q} \triangleleft \mathtt{p}?m$. We require the sender and receiver processes to be different: $\mathtt{p} \neq \mathtt{q}$. A *choice* ($\sum$) occurs at the sender role. Each branch of a choice needs to be uniquely distinguishable: $\forall i, j \in I.\, i \neq j \Rightarrow (\mathtt{q}_i, m_i) \neq (\mathtt{q}_j, m_j)$. The least fixed point operator encodes loops and we require recursion to be guarded, i.e., in $\mu t.\, G$, there is at least one message between μt and each t in G. Without loss of generality, we assume that all occurrences of t are bound and each bound variable t is distinct. As the recursion is limited to tail recursion, it is memoryless and generates regular sequences, so a global type can be interpreted as a regular language of message exchanges.

▶ **Example 2** (Load balancing). A simple load balancing scenario can be modelled with the global type:

$$\mu t.\ \mathsf{Client} \to \mathsf{Server} : \mathit{req}. + \begin{cases} \mathsf{Server} \to \mathsf{Worker}_1 : \mathit{req}.\ \mathsf{Worker}_1 \to \mathsf{Client} : \mathit{reply}.\ t \\ \mathsf{Server} \to \mathsf{Worker}_2 : \mathit{req}.\ \mathsf{Worker}_2 \to \mathsf{Client} : \mathit{reply}.\ t \end{cases}$$

The least fixed point operator μ encodes a loop in which a client sends a request to a server. The server then non-deterministically forwards the request to one of two workers. The chosen worker handles the request and replies to the client. In this protocol, the server communicates with a different worker in each branch. Figure 1a shows this example as a high-level message sequence chart (HMSC). The timeline of roles is shown with vertical lines and the messages with horizontal arrows. Different message contents are represented by different styles of arrows. ⌟



(a) Load balancing. **(b)** Variant of load balancing. **(c)** Execution.

Figure 1 Load balancing and some variant with potential for confusion exemplified by an execution.

Next, we define local types, which specify a role's view of a protocol.

▶ **Definition 3** (Local types). *The* local types *for a role* p *are defined as:*

$$L ::= 0 \;\mid\; \bigoplus_{i \in I} \mathsf{q}_i!m_i.L_i \;\mid\; \mathop{\&}_{i \in I} \mathsf{q}_i?m_i.L_i \;\mid\; \mu t.L \;\mid\; t$$

where the internal choice ($\oplus$) *and external choice* ($\&$) *both respect* $\forall i,j \in I.\ i \neq j \Rightarrow (\mathsf{q}_i, m_i) \neq (\mathsf{q}_j, m_j)$. *As for global types, we assume every recursion variable is bound, each recursion operator* (μ) *uses a different identifier* t, *and we may omit* $\oplus$ *and* $\&$ *if* $|I| = 1$.

Note that a role can send to, resp. receive from, multiple roles in a choice: we generalise $\oplus_{i\in I}\, \mathsf{q}!m_i.L_i$ of standard MSTs to $\oplus_{i\in I}\, \mathsf{q}_i!m_i.L_i$ and $\&_{i\in I}\, \mathsf{q}?m_i.L_i$ to $\&_{i\in I}\, \mathsf{q}_i?m_i.L_i$.

▶ **Example 4.** We can give the following local types for Figure 1a:

$$\begin{aligned}
\mathsf{Server}:&\quad \mu t.\, \mathsf{Client}?\mathit{req}.\, (\mathsf{Worker}_1!\mathit{req}.\, t \;\oplus\; \mathsf{Worker}_2!\mathit{req}.\, t)\\
\mathsf{Client}:&\quad \mu t.\, \mathsf{Server}!\mathit{req}.\, (\mathsf{Worker}_1?\mathit{reply}.\, t \;\&\; \mathsf{Worker}_2?\mathit{reply}.\, t)\\
\mathsf{Worker_i}:&\quad \mu t.\, \mathsf{Server}?\mathit{req}.\, \mathsf{Client}!\mathit{reply}.\, t \text{ for } i \in \{1,2\}
\end{aligned}$$

Note that their structure, i.e., having a loop with at most two options, resembles the one of the global type in Example 2. ⌟

Our goal is to define a partial *projection* operation from a given global type to a local type for each role. If the projection is defined, we expect that the type is *implementable*. We shall show that the global type of Example 2 projects to the local types in Example 4. As a consequence, the global type is implementable. Intuitively, when each role in the example executes based on its local type, they agree on a unique global path in an unrolling of the global type. We formalise projection and soundness in Section 3. We note that existing projection operations, including the ones by Hu and Yoshida [38] as well as Scalas and Yoshida [52], reject the above global type as not implementable.

Notations and Assumptions. We write $\mathbf{G}$ for the global type we try to project. When traversing the global type $\mathbf{G}$, we use G for the current term (which is a subterm of $\mathbf{G}$). To simplify the notation, we assume that the index i of a choice uniquely determines the sender and the message $\mathsf{q}_i?m_i$. Using this notation, we write $I \cap J$ to select the set of choices with identical sender and message value and $I \setminus J$ to select the alternatives present in I but not in J. When looking at send and receive events in a global setting we write $\mathsf{p} \triangleright \mathsf{q}!m$ for p sending to q and $\mathsf{q} \triangleleft \mathsf{p}?m$ for q receiving from p.

In later definitions, we unfold the recursion in types. We could get the unfolding through a congruence relation. However, this requires dealing with infinite structures, which makes some definitions not effective. Instead, we precompute the map from each recursion variable t to its unfolding. For a given global type, let $\mathit{get}\mu$ be a function that returns a map from t to G for each subterm $\mu t.\, G$. Recall, each t in a type is different. $\mathit{get}\mu$ is defined as follows:

$$get\mu(0) := [\,] \qquad get\mu(t) := [\,] \qquad get\mu(\mu t.G) := [t \mapsto G] \cup get\mu(G)$$

$$get\mu(\sum_{i\in I} \mathtt{p}\to\mathtt{q}_i : m_i.G_i) := \bigcup_{i\in I} get\mu(G_i)$$

We write $get\mu_{\mathbf{G}}$ as shorthand for the map returned by $get\mu(\mathbf{G})$.

2.2 Generalised Projection and Merge

We now define a partial *projection* operation that projects a global type on to each role. The projection on to a role $\mathtt{r}$ is a local type and keeps only $\mathtt{r}$'s actions. Intuitively, it gives the "local view" of message exchanges performed by $\mathtt{r}$. While projecting, non-determinism may arise due to choices that $\mathtt{r}$ does not observe directly. In this case, the different branches are merged using a partial *merge* operator ($\sqcap$). The merge operator checks that a role, which has not yet learned the outcome of a choice, only performs actions that are allowed in all possible branches. The role can perform branch-specific actions after it has received a message that resolves the choice. For a role that is agnostic to the choice, i.e., behaves the same on all the branches, the merge allows the role to proceed as if the choice does not exist.

So far, the idea follows standard asynchronous MSTs. What distinguishes our new projection operator from prior ones (e.g., [38, 52]), is that we allow a role to learn which branch has been taken through messages received from different senders. This generalisation is non-trivial. When limiting the reception to messages from a single role, one can rely on the FIFO order provided by the corresponding channel. However, messages coming from different sources are only partially ordered. Thus, unlike previous approaches, our merge operator looks at the result of a *causality analysis* on the global type to make sure that this partial ordering cannot introduce any confusion.

▶ **Example 5** (Intricacies of generalising projection)**.** We demonstrate that a straightforward generalisation of existing projection operators can lead to unsoundness. Consider a *naive* projection that merges branches with internal choice if they are equal, and for receives, simply always merges external choices – also from different senders. In addition, it removes empty loops. For Figure 1a, this naive projection yields the expected local types presented in Example 4. We show that naive projection can be unsound. Figure 1b shows a variant of load balancing, for which naive projection yields the following local types:

$$\begin{aligned}
\mathsf{Server}: &\quad \mu t.\,\mathsf{Client}?req.\,(\mathsf{Worker}_1!req.\,t \;\oplus\; \mathsf{Worker}_2!req.\,t)\\
\mathsf{Client}: &\quad \mu t.\,\mathsf{Server}!req.\,(\mathsf{Worker}_1?reply.\,\mathsf{Worker}_2?reply.\,t \;\&\; \mathsf{Worker}_2?reply.\,t)\\
\mathsf{Worker}_1: &\quad \mu t.\,\mathsf{Server}?req.\,\mathsf{Worker}_2!req.\,t\\
\mathsf{Worker}_2: &\quad \mu t.\,(\mathsf{Worker}_1?req.\,\mathsf{Client}{\triangleright}!reply.\,t \;\&\; \mathsf{Server}?req.\,\mathsf{Client}{\triangleright}!reply.\,t)
\end{aligned}$$

Unfortunately, the global type is not implementable. The problem is that, for the *Client*, the two messages on its left branch are not causally related. Consider the execution prefix in Figure 1c which is not specified in the global type. The Server decided to first take the left (L) and then the right (R) branch. For Server, the order LR is obvious from its events and the same applies for Worker$_2$. For Worker$_1$, every possible order R^*LR^* is plausible as it does not have events in the right branch. Since LR belongs to the set of plausible orders, there is no confusion. Now, the messages from the two workers to the client are independent and, therefore, can be received in any order. If the client receives *Worker$_2$?reply* first, then its local view is consistent with the choice RL as the order of branches. This can lead to confusion and, thus, execution prefixes which are not specified in the global type. ⌟

We shall now define our generalised projection operation. To identify confusion as above, we keep track of causality between messages. We determine what messages a role *could* receive at a given point in the global type through an *available messages* analysis. Tracking causality needs to be done at the level of the global type. We look for chains of dependent messages and we also need to unfold loops. Fortunately, since we only check for the presence or absence of some messages, it is sufficient to unfold each recursion at most once.

Projection and Interferences from Independent Messages. The challenge of projecting a global type lies in resolving the non-determinism introduced by having only the endpoint view. Example 5 shows that in order to decide if a choice is safe, we need to know which messages can arrive at the same time. To enable this, we annotate local types with the messages that could be received at that point in the protocol. We call these *availability annotated local types* and write them as $AL = \langle L, \mathit{Msg}\rangle$ where L is a local type and Msg is a set of messages. This signifies that when a role has reached AL, the messages in Msg can be present in the communication channels. We annotate types using the grammar for local types (Definition 3), where each subterm is annotated. To recover a local type, we erase the annotation, i.e., recursively replace each $AL = \langle L, \mathit{Msg}\rangle$ by L. The projection internally uses annotated types.

The projection of $\mathbf{G}$ on to $\mathtt{r}$, written $\mathbf{G}{\restriction}_{\mathtt{r}}$, traverses $\mathbf{G}$ to erase the operations that do not involve $\mathtt{r}$. During this phase, we also compute the messages that $\mathtt{r}$ may receive. The function $\operatorname{avail}(B, T, G)$ computes the set of messages that other roles can send while $\mathtt{r}$ has not yet learned the outcome of the choice. This set depends on B, the set of *blocked* roles, i.e., the roles which are waiting to receive a message and hence cannot move; T, the set of recursion variables we have already visited; and G, the subterm in $\mathbf{G}$ at which we compute the available messages. We defer the definition of $\operatorname{avail}(B, T, G)$ to later in this section.

Empty Paths Elimination. When projecting, there may be paths and loops where a role neither sends nor receives a message, e.g., the right loop in Example 2 for Worker$_1$. Such paths can be removed during projection. Even if conceptually simple, the notational overhead impedes understandability of how our message availability analysis is used. Therefore, we first focus on the message availability analysis and define a projection operation that does not account for empty paths elimination. After defining the merge operator $\sqcap$, we give the full definition of our generalised projection operation.

▶ **Definition 6** (Projection without empty paths elimination). *The* projection without empty paths elimination $\mathbf{G}{\restriction}_{\mathtt{r}}$ *of a global type* $\mathbf{G}$ *on to a role* $\mathtt{r} \in \mathcal{P}$ *is an availability annotated local type inductively defined as:*

$$t{\restriction}_{\mathtt{r}} := \langle t, \operatorname{avail}(\{\mathtt{r}\}, \{t\}, \mathit{get}\mu_{\mathbf{G}}(t))\rangle \qquad\qquad 0{\restriction}_{\mathtt{r}} := \langle 0, \emptyset\rangle$$

$$(\mu t.G){\restriction}_{\mathtt{r}} := \begin{cases} \langle \mu t.(G{\restriction}_{\mathtt{r}}), \operatorname{avail}(\{\mathtt{r}\}, \{t\}, G)\rangle & \textit{if } G{\restriction}_{\mathtt{r}} \neq \langle t, \_\rangle \\ \langle 0, \emptyset\rangle & \textit{otherwise} \end{cases}$$

$$\Big(\textstyle\sum_{i\in I} \mathtt{p}\to\mathtt{q}_i : m_i.G_i\Big){\restriction}_{\mathtt{r}} := \begin{cases} \langle \oplus_{i\in I}\, \mathtt{q}_i!m_i.(G_i{\restriction}_{\mathtt{r}}), \bigcup_{i\in I} \operatorname{avail}(\{\mathtt{q}_i, \mathtt{r}\}, \emptyset, G_i)\rangle & \textit{if } \mathtt{r} = \mathtt{p} \\ \sqcap \begin{pmatrix} \langle \&_{i\in I_{[=\mathtt{r}]}}\, \mathtt{p}?m_i.(G_i{\restriction}_{\mathtt{r}}), \bigcup_{i\in I_{[=\mathtt{r}]}} \operatorname{avail}(\{\mathtt{r}\}, \emptyset, G_i)\rangle \\ \sqcap_{i\in I_{[\neq\mathtt{r}]}}\, G_i{\restriction}_{\mathtt{r}} \end{pmatrix} & \textit{otherwise} \\ \textit{where } I_{[=\mathtt{r}]} := \{i \in I \mid \mathtt{q}_i = \mathtt{r}\} \textit{ and } I_{[\neq\mathtt{r}]} := \{i \in I \mid \mathtt{q}_i \neq \mathtt{r}\} & \end{cases}$$

A global type $\mathbf{G}$ *is said to be* projectable *if* $\mathbf{G}{\restriction}_{\mathtt{r}}$ *is defined for every* $\mathtt{r} \in \mathcal{P}$.

Projection erases events not relevant to $\mathtt{r}$ by a recursive traversal of the global type; however, at a choice not involving $\mathtt{r}$, it has to ensure that either $\mathtt{r}$ is indifferent to the outcome of the choice or it indirectly receives enough information to distinguish the outcome.

This is managed by the merge operator $\sqcap$ and the use of available messages. The merge operator takes as arguments a sequence of availability annotated local types. Our merge operator generalises the full merge by Scalas and Yoshida [52]. When faced with choice, it only merges receptions that cannot interfere with each other. For the sake of clarity, we define only the binary merge. As the operator is commutative and associative, it generalises to a set of branches I. When I is a singleton, the merge just returns that one branch.

▶ **Definition 7** (Merge operator $\sqcap$). *Let $\langle L_1, Msg_1\rangle$ and $\langle L_2, Msg_2\rangle$ be availability annotated local types for a role $\mathtt{r}$. $\langle L_1, Msg_1\rangle \sqcap \langle L_2, Msg_2\rangle$ is defined by cases, as follows:*

- $\langle L_1, Msg_1 \cup Msg_2\rangle$ *if* $L_1 = L_2$
- $\langle \mu t_1.(AL_1 \sqcap AL_2[t_2/t_1]), Msg_1 \cup Msg_2\rangle$ *if* $L_1 = \mu t_1.AL_1,\ L_2 = \mu t_2.AL_2$
- $\langle \oplus_{i\in I}\, \mathtt{q}_i!m_i.(AL_{1,i} \sqcap AL_{2,i}), Msg_1 \cup Msg_2\rangle$ *if* $\begin{cases} L_1 = \oplus_{i\in I}\, \mathtt{q}_i!m_i.AL_{1,i}, \\ L_2 = \oplus_{i\in I}\, \mathtt{q}_i!m_i.AL_{2,i} \end{cases}$
- $$\begin{array}{ll} \langle & \&_{i\in I\setminus J}\, \mathtt{q}_i?m_i.AL_{1,i} \quad \& \\ & \&_{i\in I\cap J}\, \mathtt{q}_i?m_i.(AL_{1,i} \sqcap AL_{2,i}) \quad \& \\ & \&_{i\in J\setminus I}\, \mathtt{q}_i?m_i.AL_{2,i} \quad , \\ & Msg_1 \cup Msg_2 \quad \rangle \end{array} \quad \textit{if} \begin{cases} L_1 = \&_{i\in I}\, \mathtt{q}_i?m_i.AL_{1,i}, \\ L_2 = \&_{i\in J}\, \mathtt{q}_i?m_i.AL_{2,i}, \\ \forall i \in I \setminus J.\, \mathtt{r} \triangleleft \mathtt{q}_i?m_i \notin Msg_2, \\ \forall i \in J \setminus I.\, \mathtt{r} \triangleleft \mathtt{q}_i?m_i \notin Msg_1 \end{cases}$$

When no condition applies, the merge and, thus, the projection are undefined.[1]

The important case of the merge is the external choice. Here, when a role can potentially receive a message that is unique to a branch, it checks that the message cannot be available in another branch so actually being able to receive this message uniquely determines which branch was taken by the role to choose. For the other cases, a role can postpone learning the branch as long as the actions on both branches are the same.

Adding Empty Paths Elimination. The preliminary version of projection requires every role to have at least one event in each branch of a loop and, thus, rejects examples where a role has no event in some loop branch. Such paths can be eliminated. However, determining such empty paths cannot be done on the level of the merge operator but only when projecting. To this end, we introduce an additional parameter E for the generalised projection: E contains those variables t for which $\mathtt{r}$ has not observed any message send or receive event since μt.

▶ **Definition 8** (Generalised projection – with empty paths elimination). *The* projection $\mathbf{G}{\restriction}^{E}_{\mathtt{r}}$ *of a global type $\mathbf{G}$ on to a role $\mathtt{r} \in \mathcal{P}$ is an availability annotated local type which is inductively defined as follows:*

$$t{\restriction}^{E}_{\mathtt{r}} := \langle t, \mathrm{avail}(\{R\}, \{t\}, get\mu_{\mathbf{G}}(t))\rangle \qquad 0{\restriction}^{E}_{\mathtt{r}} := \langle 0, \emptyset\rangle$$

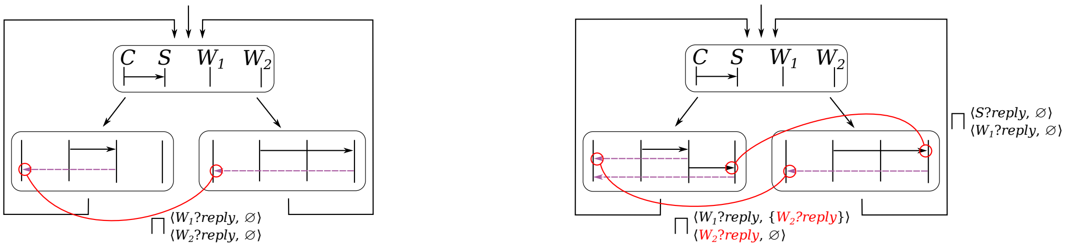
$$(\mu t.G){\restriction}^{E}_{\mathtt{r}} := \begin{cases} \langle \mu t.(G{\restriction}^{E\cup\{t\}}_{\mathtt{r}}), \mathrm{avail}(\{R\}, \{t\}, G)\rangle & \textit{if } G{\restriction}^{E\cup\{t\}}_{\mathtt{r}} \neq \langle t, \_\rangle \\ \langle 0, \emptyset\rangle & \textit{otherwise} \end{cases}$$

$$\left(\textstyle\sum_{i\in I} \mathtt{p}{\to}\mathtt{q}_i : m_i.G_i\right){\restriction}^{E}_{\mathtt{r}} := \begin{cases} \langle \oplus_{i\in I}\, \mathtt{q}_i!m_i.(G_i{\restriction}^{\emptyset}_{\mathtt{r}}), \bigcup_{i\in I} \mathrm{avail}(\{\mathtt{q}_i, \mathtt{r}\}, \emptyset, G_i)\rangle & \textit{if } \mathtt{r} = \mathtt{p} \\ \sqcap \left(\begin{array}{l} \langle \&_{i\in I_{[=\mathtt{r}]}}\, \mathtt{p}?m_i.(G_i{\restriction}^{\emptyset}_{\mathtt{r}}), \bigcup_{i\in I_{[=\mathtt{r}]}} \mathrm{avail}(\{\mathtt{r}\}, \emptyset, G_i)\rangle \\ \sqcap_{i\in I_{[\neq\mathtt{r}]} \wedge \forall t\in E.\, G_i{\restriction}^{E}_{\mathtt{r}} \neq \langle t,\_\rangle}\, G_i{\restriction}^{E}_{\mathtt{r}} \end{array} \right) & \textit{otherwise} \end{cases}$$

$$\textit{where } I_{[=\mathtt{r}]} := \{i \in I \mid \mathtt{q}_i = \mathtt{r}\} \textit{ and } I_{[\neq\mathtt{r}]} := \{i \in I \mid \mathtt{q}_i \neq \mathtt{r}\}$$

Since the merge operator $\sqcap$ is partial, the projection may be undefined. We use $\mathbf{G}{\restriction}_{\mathtt{r}}$ as shorthand for $\mathbf{G}{\restriction}^{\emptyset}_{\mathtt{r}}$ and only consider the generalised projection with empty paths elimination from now on. A global type $\mathbf{G}$ is called projectable *if $\mathbf{G}{\restriction}_{\mathtt{r}}$ is defined for every role $\mathtt{r} \in \mathcal{P}$.*

[1] When we use the n-ary notation $\sqcap_{i\in I}$ and $|I| = 0$, we implicitly omit this part. Note that this can only happen if $\mathtt{r}$ is the receiver among all branches for some choice so there is either another local type to merge with, or the projection is undefined anyway.



(a) Merging for Figure 1a. **(b)** Merging for Figure 1b.

Figure 2 Availability annotated types for merging on the two examples. The red lines connect the receptions that get merged during projection. The annotations only show the receiver's messages.

We highlight the differences for the empty paths elimination. Recall that E contains all recursion variables from which the role $\mathtt{r}$ has not encountered any events. To guarantee this, for the case of recursion $\mu t.\, G$, the (unique) variable t is added to the current set E, while the parameter turns to the empty set $\emptyset$ as soon as $\mathtt{r}$ encounters an event. The previous steps basically constitute the necessary bookkeeping. The actual elimination is achieved with the condition $\forall t \in E.\, G_i{\restriction}^E_{\mathtt{r}} \neq \langle t, \_ \rangle$ which filters all branches without events of role $\mathtt{r}$.

Other works [38, 17] achieve this with *connecting actions*, marking non-empty paths. Like classical MSTs, we do not include such explicit actions. Still, we can automatically eliminate such paths in contrast to previous work.

Computing Available Messages. Finally, the function avail is computed recursively:

$$\mathrm{avail}(B, T, 0) := \emptyset$$

$$\mathrm{avail}(B, T, \mu t.G) := \mathrm{avail}(B, T \cup \{t\}, G)$$

$$\mathrm{avail}(B, T, t) := \begin{cases} \emptyset & \text{if } t \in T \\ \mathrm{avail}(B, T \cup \{t\}, get\mu_{\mathbf{G}}(t)) & \text{if } t \notin T \end{cases}$$

$$\mathrm{avail}(B, T, \textstyle\sum_{i \in I} \mathtt{p} \to \mathtt{q}_i : m_i.G_i) := \begin{cases} \bigcup_{i \in I, m \in \mathcal{V}} (\mathrm{avail}(B, T, G_i) \setminus \{\mathtt{q}_i \triangleleft \mathtt{p}?m\}) \cup \{\mathtt{q}_i \triangleleft \mathtt{p}?m_i\} & \text{if } \mathtt{p} \notin B \\ \bigcup_{i \in I} \mathrm{avail}(B \cup \{\mathtt{q}_i\}, T, G_i) & \text{if } \mathtt{p} \in B \end{cases}$$

Since all channels are FIFO, we only keep the first possible message in each channel. The fourth case discards messages not at the head of the channel.

Our projection is different from the one of Scalas and Yoshida [52], not just because our syntax is more general. It also represents a shift in paradigm. In their work, the full merge works only on local types. No additional knowledge is required. This is possible because their type system limits the flexibility of communication. Since we allow more flexible communication, we need to keep some information, in form of available messages, about the possible global executions for the merge operator.

▶ **Example 9.** Let us explain how our projection operator catches the problem in $\mathbf{G}{\restriction}_{\mathit{Client}}$ of Figure 1b. Figure 2 shows the function of available messages during the projection for Figure 1a and Figure 1b. In Figure 2a, the messages form chains, i.e., except for the role making the choice, a role only sends in reaction to another message. Therefore, only a single message is available at each reception and the protocol is projectable. On the other hand, in Figure 2b both replies are available and, therefore, the protocol is not projectable.

Here are the details of the projection for Figure 2b. If not needed, we omit the availability annotations for readability. Recall that Client receives *reply* from Worker_2 in the left branch, which is also present in the right branch. Let us denote the two branches as follows:

$$G_1 := \mathsf{Server} \to \mathsf{Worker}_1 : \mathit{req}.\ \mathsf{Worker}_1 \to \mathsf{Worker}_2 : \mathit{req}.\ \mathsf{Worker}_2 \to \mathsf{Client} : \mathit{reply}.\ t, \text{ and}$$

$$G_2 := \mathsf{Server} \to \mathsf{Worker}_2 : \mathit{req}.\ \mathsf{Worker}_2 \to \mathsf{Client} : \mathit{reply}.\ t$$

Since the first message in G_1 does not involve Client, the projection descends and we compute:

$$\begin{aligned} \mathbf{G}{\restriction}_{\mathsf{Client}} \quad &= \langle \mu t.(\langle \mathsf{Worker}_1?\mathit{reply}.\,(G_1'{\restriction}_{\mathsf{Client}}),\ \mathrm{avail}(\{\mathsf{Client}\},\emptyset,G_1')\rangle \\ &\qquad\quad \sqcap \langle \mathsf{Worker}_2?\mathit{reply}.\,(G_2'{\restriction}_{\mathsf{Client}}),\ \mathrm{avail}(\{\mathsf{Client}\},\emptyset,G_2')\rangle), \_\rangle \\ \text{where} \quad & G_1' = \mathsf{Worker}_1 \to \mathsf{Worker}_2 : \mathit{req}.\ \mathsf{Worker}_2 \to \mathsf{Client} : \mathit{reply}.\ t \text{ and } G_2' = t. \end{aligned}$$

For this, we compute $\mathrm{avail}(\{\mathsf{Client}\},\emptyset,G_1') = \emptyset$ and $\mathrm{avail}(\{\mathsf{Client}\},\emptyset,G_2') = \{\mathsf{Worker}_2 \triangleleft \mathsf{Worker}_1?\mathit{req}, \mathsf{Client} \triangleleft \mathsf{Worker}_2?\mathit{reply}\}$ and see that the conditions are not satisfied. Indeed, $\mathsf{Client} \triangleleft \mathsf{Worker}_2?\mathit{reply} \in \mathrm{avail}(\{\mathsf{Client}\},\emptyset,G_2')$. Thus, the projection is undefined. ⌟

3 Type Soundness

We now show a soundness theorem for generalised projection; roughly, a projectable global type can be implemented by communicating state machines in a distributed way. Our proof uses automata-theoretic techniques from the theory of MSCs. We assume familiarity with the basics of formal languages.

As our running example shows, a protocol implementation often cannot enforce the event ordering specified in the type but only a weaker order. In this section, we capture both notions through a type language and an execution language.

3.1 Type Languages

A *state machine* $A = (Q,\Sigma,\delta,q_0,F)$ consists of a finite set Q of states, an alphabet Σ, a transition relation $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$, an initial state $q_0 \in Q$, and a set $F \subseteq Q$ of final states. We write $q \xrightarrow{x} q'$ for $(q,x,q') \in \delta$. We define the runs and traces in the standard way. A run is maximal if it is infinite or if it ends at a final state. The *language* $\mathcal{L}(A)$ is the set of (finite or infinite) maximal traces. The projection $A{\Downarrow}_\Delta$ of a state machine is its projection to a sub-alphabet $\Delta \subseteq \Sigma$ obtained by replacing all letters in $\Sigma \setminus \Delta$ with ε-transitions. It accepts the language $\mathcal{L}(A){\Downarrow}_\Delta = \{w{\Downarrow}_\Delta \mid w \in \mathcal{L}(A)\}$.

▶ **Definition 10** (Type language for global types). *The semantics of a global type* $\mathbf{G}$ *is given as a regular language. We construct a state machine* $\mathsf{GAut}(\mathbf{G})$ *using an auxiliary state machine* $M(\mathbf{G})$*. First, we define* $M(\mathbf{G}) = (Q_{M(\mathbf{G})}, \Sigma_{sync}, \delta_{M(\mathbf{G})}, q_{0M(\mathbf{G})}, F_{M(\mathbf{G})})$ *where*

- $Q_{M(\mathbf{G})}$ *is the set of all syntactic subterms in* $\mathbf{G}$ *together with the term* 0*,*
- $\Sigma_{sync} = \{\mathsf{p} \to \mathsf{q} : m \mid \mathsf{p},\mathsf{q} \in \mathcal{P} \text{ and } m \in \mathcal{V}\}$*,*
- $\delta_{M(\mathbf{G})}$ *is the smallest set containing* $(\sum_{i \in I} \mathsf{p} \to \mathsf{q}_i : m_i.G_i, \mathsf{p} \to \mathsf{q}_i : m_i, G_i)$ *for each* $i \in I$*, as well as* $(\mu t.G', \varepsilon, G')$ *and* $(t, \varepsilon, \mu t.G')$ *for each subterm* $\mu t.G'$ *of* $\mathbf{G}$*,*
- $q_{0M(\mathbf{G})} = \mathbf{G}$ *and* $F_{M(\mathbf{G})} = \{0\}$*.*

Next, we expand each message $\mathsf{p} \to \mathsf{q} : m$ *into two events,* $\mathsf{p} \triangleright \mathsf{q}!m$ *followed by* $\mathsf{q} \triangleleft \mathsf{p}?m$*. We define* $\mathsf{GAut}(\mathbf{G}) = (Q_{\mathsf{GAut}(\mathbf{G})}, \Sigma_{\mathsf{GAut}(\mathbf{G})}, \delta_{\mathsf{GAut}(\mathbf{G})}, q_{0\mathsf{GAut}(\mathbf{G})}, F_{\mathsf{GAut}(\mathbf{G})})$ *as follows:*

- $Q_{\mathsf{GAut}(\mathbf{G})} = Q_{M(\mathbf{G})} \cup (Q_{M(\mathbf{G})} \times \Sigma_{sync} \times Q_{M(\mathbf{G})})$*,*
- $\Sigma_{\mathsf{GAut}(\mathbf{G})} = \{\mathsf{p} \triangleright \mathsf{q}!m \mid \mathsf{p},\mathsf{q} \in \mathcal{P}, m \in \mathcal{V}\} \cup \{\mathsf{q} \triangleleft \mathsf{p}?m \mid \mathsf{p},\mathsf{q} \in \mathcal{P}, m \in \mathcal{V}\}$*,*
- $\delta_{\mathsf{GAut}(\mathbf{G})}$ *is the smallest set containing the transitions* $(s, \mathsf{p} \triangleright \mathsf{q}!m, (s, \mathsf{p} \to \mathsf{q} : m, s'))$ *and* $((s, \mathsf{p} \to \mathsf{q} : m, s'), \mathsf{q} \triangleleft \mathsf{p}?m, s'))$ *for each transition* $(s, \mathsf{p} \to \mathsf{q} : m, s') \in \delta_{M(\mathbf{G})}$*,*
- $q_{0\mathsf{GAut}(\mathbf{G})} = q_{0M(\mathbf{G})}$ *and* $F_{\mathsf{GAut}(\mathbf{G})} = F_{M(\mathbf{G})}$*.*

The type language $\mathcal{L}(\mathbf{G})$ *of a global type* $\mathbf{G}$ *is given by* $\mathcal{L}(\mathsf{GAut}(\mathbf{G}))$*.*

▶ **Definition 11** (Type language for local types). *Given a local type L for $\mathtt{p}$, we construct a state machine $\mathsf{LAut}(L) = (Q, \Sigma_{\mathtt{p}}, \delta, q_0, F)$ where*

- *Q is the set of all syntactic subterms in L,*
- *$\Sigma_{\mathtt{p}} = \{\mathtt{p} \triangleright \mathtt{q}!m \mid \mathtt{q} \in \mathcal{P}, m \in \mathcal{V}\} \cup \{\mathtt{p} \triangleleft \mathtt{q}?m \mid \mathtt{p}, \mathtt{q} \in \mathcal{P}, m \in \mathcal{V}\}$,*
- *δ is the smallest set containing $(\oplus_{i\in I}\, \mathtt{q}_i!m_i.L_i, \mathtt{p} \triangleright \mathtt{q}_i!m_i, L_i)$ and $(\&_{i\in I}\, \mathtt{q}_i?m_i.L_i, \mathtt{p} \triangleleft \mathtt{q}_i?m_i, L_i)$ for each $i \in I$, as well as $(\mu t.L', \varepsilon, L')$ and $(t, \varepsilon, \mu t.L')$ for each $\mu t.L'$ in L,*
- *$q_0 = L$ and $F = \{0\}$ if 0 is a subterm of L, and empty otherwise.*

We define the type language *of L as language of this automaton: $\mathcal{L}(L) = \mathcal{L}(\mathsf{LAut}(L))$.*

3.2 Implementability

An *implementation* consists of a set of state machines, one per role, communicating with each other through asynchronous messages and pairwise FIFO channels. We use communicating state machines (CSMs) [10] as our formal model. A CSM $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ consists of a set of state machines $A_{\mathtt{p}}$, one for each $\mathtt{p} \in \mathcal{P}$ over the alphabet of message sends and receives. Communication between machines happens asynchronously through FIFO channels. The semantics of a CSM is a language $\mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}})$ of maximal traces over the alphabet of message sends and receives satisfying the FIFO condition on channels. A CSM is *deadlock free* if every trace can be extended to a maximal trace. We omit the (standard) formal definition of CSMs (see Appendix A for details).

Indistinguishability Relation. In the type language of a global type, every send event is always immediately succeeded by its receive event. However, in a CSM, other independent events may occur between the send and the receipt and there is no way to force the order specified by the type language. To capture this phenomenon formally, we define a family of *indistinguishability relations* $\sim_i \subseteq \Sigma^* \times \Sigma^*$, for $i \geq 0$ and $\Sigma = \Sigma_{\mathsf{GAut}(\mathbf{G})}$, as follows. For all $w \in \Sigma^*$, we have $w \sim_0 w$. For $i = 1$, we define:

(1) If $\mathtt{p} \neq \mathtt{r}$, then $w.\mathtt{p} \triangleright \mathtt{q}!m.\mathtt{r} \triangleright \mathtt{s}!m'.u \;\sim_1\; w.\mathtt{r} \triangleright \mathtt{s}!m'.\mathtt{p} \triangleright \mathtt{q}!m.u$.

(2) If $\mathtt{q} \neq \mathtt{s}$, then $w.\mathtt{q} \triangleleft \mathtt{p}?m.\mathtt{s} \triangleleft \mathtt{r}?m'.u \;\sim_1\; w.\mathtt{s} \triangleleft \mathtt{r}?m'.\mathtt{q} \triangleleft \mathtt{p}?m.u$.

(3) If $\mathtt{p} \neq \mathtt{s} \wedge (\mathtt{p} \neq \mathtt{r} \vee \mathtt{q} \neq \mathtt{s})$, then $w.\mathtt{p} \triangleright \mathtt{q}!m.\mathtt{s} \triangleleft \mathtt{r}?m'.u \;\sim_1\; w.\mathtt{s} \triangleleft \mathtt{r}?m'.\mathtt{p} \triangleright \mathtt{q}!m.u$.

(4) If $|w\Downarrow_{\mathtt{p}\triangleright\mathtt{q}!\_}| > |w\Downarrow_{\mathtt{q}\triangleleft\mathtt{p}?\_}|$, then $w.\mathtt{p} \triangleright \mathtt{q}!m.\mathtt{q} \triangleleft \mathtt{p}?m'.u \;\sim_1\; w.\mathtt{q} \triangleleft \mathtt{p}?m'.\mathtt{p} \triangleright \mathtt{q}!m.u$.

We refer to the proof of Lemma 21 in Appendix B for further details on the conditions for swapping events. Let w, w', w'' be sequences of events s.t. $w \sim_1 w'$ and $w' \sim_i w''$ for some i. Then, $w \sim_{i+1} w''$. We define $w \sim u$ if $w \sim_n u$ for some n. It is straightforward that $\sim$ is an equivalence relation. Define $u \preceq_\sim v$ if there is $w \in \Sigma^*$ such that $u.w \sim v$. Observe that $u \sim v$ iff $u \preceq_\sim v$ and $v \preceq_\sim u$. To extend $\sim$ to infinite words, we follow the approach of Gastin [27]. For infinite words $u, v \in \Sigma^\omega$, we define $u \preceq_\sim^\omega v$ if for each finite prefix u' of u, there is a finite prefix v' of v such that $u' \preceq_\sim v'$. Define $u \sim v$ iff $u \preceq_\sim^\omega v$ and $v \preceq_\sim^\omega u$.

We lift the equivalence relation $\sim$ on words to languages:

For a language Λ, we define $\mathcal{C}^\sim(\Lambda) = \left\{ w' \mid \bigvee \begin{array}{l} w' \in \Sigma^* \wedge \exists w \in \Sigma^*.\ w \in \Lambda \text{ and } w' \sim w \\ w' \in \Sigma^\omega \wedge \exists w \in \Sigma^\omega.\ w \in \Lambda \text{ and } w' \preceq_\sim^\omega w \end{array} \right\}$.

For the infinite case, we take the downward closure w.r.t. $\preceq_\sim^\omega$. Unlike [27, Definition 2.1], our closure operator is asymmetric. Consider the protocol $(\mathtt{p} \triangleright \mathtt{q}!m.\ \mathtt{q} \triangleleft \mathtt{p}?m)^\omega$. Since we do not make any fairness assumption on scheduling, we need to include in the closure the execution where only the sender is scheduled, i.e., $(\mathtt{p} \triangleright \mathtt{q}!m)^\omega \preceq_\sim^\omega (\mathtt{p} \triangleright \mathtt{q}!m.\ \mathtt{q} \triangleleft \mathtt{p}?m)^\omega$.

▶ **Example 12** (Indistinguishability relation $\sim$ by examples)**.** The four rules for $\sim_1$ present conditions under which two adjacent events in an execution (prefix) can be swapped. These conditions are designed such that they characterise possible changes in an execution (prefix) which cannot be recognised by any CSM. To be precise, if w is recognised by some CSM $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ and $w' \sim_1 w$ holds, then w' is also recognised by $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$. In this example, we illustrate the intuition behind these rules.

For the remainder of this example, the *active role* of an event is the receiver of a receive event and the sender of a send event. Visually, the active role is always the first role in an event. In addition, we assume that variables do not alias, i.e., two roles or messages with different names are different.

Two send events (or two receive events) can be swapped if the active roles are distinct because there cannot be any dependency between two such events which do occur next to each other in an execution. For send events, the 1st rule, thus, admits $\mathtt{p} \triangleright \mathtt{r}!m.\ \mathtt{q} \triangleright \mathtt{r}!m \sim_1 \mathtt{q} \triangleright \mathtt{r}!m.\ \mathtt{p} \triangleright \mathtt{r}!m$ even though the receiver is the same. In contrast, the corresponding receive events cannot be swapped: $\mathtt{r} \triangleleft \mathtt{p}?m.\ \mathtt{r} \triangleleft \mathtt{q}?m \not\sim_1 \mathtt{r} \triangleleft \mathtt{q}?m.\ \mathtt{r} \triangleleft \mathtt{p}?m$. Note that the 1st rule is the only one with which two send events can be swapped while the 2nd rule is the only one for receive events so indeed no rule applies for the last case.

The 3rd rule allows one send and one receive event to be swapped if either both senders or both receivers are different – in addition to the requirement that both active roles are different. For instance, it admits $\mathtt{p} \triangleright \mathtt{r}!m.\ \mathtt{q} \triangleleft \mathtt{r}?m \sim_1 \mathtt{q} \triangleleft \mathtt{r}?m.\ \mathtt{p} \triangleright \mathtt{r}!m$. However, it does not admit two swap $\mathtt{p} \triangleright \mathtt{q}!m.\ \mathtt{q} \triangleleft \mathtt{p}?m \not\sim_1 \mathtt{q} \triangleleft \mathtt{p}?m.\ \mathtt{p} \triangleright \mathtt{q}!m$. This is reasonable since the send event could be the one which emits m in the corresponding channel. In this execution prefix, this is in fact the case since there have been no events before, but in general one needs to incorporate the context to understand whether this is the case. The 4th rule does this and therefore admits swapping the same events when appended to some prefix: $\mathtt{p} \triangleright \mathtt{q}!m.\mathtt{p} \triangleright \mathtt{q}!m.\ \mathtt{q} \triangleleft \mathtt{p}?m \not\sim_1 \mathtt{p} \triangleright \mathtt{q}!m.\mathtt{q} \triangleleft \mathtt{p}?m.\ \mathtt{p} \triangleright \mathtt{q}!m$. Then, the FIFO order of channels ensures that the first message will be received first and the 2nd send event can happen after the reception of the 1st message. ⌟

▶ **Example 13** (Load balancing revisited)**.** Let us consider the execution with confusion in Figure 1c. Compared to a synchronous execution, we need to delay the reception $C \triangleleft W_1?\mathit{reply}$ to come after the first $C \triangleleft W_2?\mathit{reply}$. Using the 2nd and 3rd cases of $\sim$ we can move $C \triangleleft W_1?\mathit{reply}$ across the communications between the two workers. Finally, we use the 3rd case again to swap $C \triangleleft W_1?\mathit{reply}$ and $W_2 \triangleright C!\mathit{reply}$ to get the desired sequence. ⌟

This example shows that $\sim$ does not change the order of send and receive events of a single role. Thus, if $w, w' \in \Sigma_\mathtt{p}^\infty$, then $w \sim w'$ iff $w = w'$. Hence, any language over the message alphabet of a single role is (trivially) closed under $\sim$. Further, two runs of a CSM on w and w' with $w \sim w'$ end in the same configuration.

Execution Languages. For a global type $\mathbf{G}$, the above discussion implies that any implementation $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ can at most achieve that $\mathcal{L}(\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}) = \mathcal{C}^\sim(\mathcal{L}(\mathbf{G}))$. This is why we call $\mathcal{C}^\sim(\mathcal{L}(\mathbf{G}))$ the *execution language* of $\mathbf{G}$. One might also call $\mathcal{C}^\sim(\mathcal{L}(L))$ of a local type L an execution language, however, since $\sim$ does not swap any events on the same role, the type language and execution language are equivalent.

▶ **Definition 14** (Implementability)**.** *A global type* $\mathbf{G}$ *is said to be* implementable *if there exists a CSM* $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ *s.t. (i) [protocol fidelity]* $\mathcal{L}(\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}) = \mathcal{C}^\sim(\mathcal{L}(\mathbf{G}))$*, and (ii) [deadlock freedom]* $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ *is deadlock free. We say that* $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ *implements* $\mathbf{G}$*.*

3.3 Type Soundness: Projectability implies Implementability

The projection operator preserves the local order of events for every role and does not remove any possible event. Therefore, we can conclude that, for each role, the projected language of the global type is subsumed by the language of the projection.

▶ **Proposition 15.** *For every projectable* $\mathbf{G}$*, role* $\mathtt{r} \in \mathcal{P}$*, run with trace* w *in* $\mathsf{GAut}(\mathbf{G})\Downarrow_{\Sigma_\mathtt{r}}$*, there is a run with trace* w *in* $\mathsf{LAut}(\mathbf{G}{\restriction_\mathtt{r}})$*. Therefore, it holds that* $\mathcal{L}(\mathbf{G})\Downarrow_{\Sigma_\mathtt{r}} \subseteq \mathcal{L}(\mathbf{G}{\restriction_\mathtt{r}})$*.*

The previous result shows that the projection does not remove behaviours. Now, we also need to show that it does not add unwanted behaviours. The main result is the following.

▶ **Theorem 16.** *If a global type* $\mathbf{G}$ *is projectable, then* $\mathbf{G}$ *is implementable.*

The complete proof can be found in the extended version [46]. Here, we give a brief summary of the main ideas. To show that a projectable global type is implemented by its projections, we need to show that the projection preserves behaviours, does not add behaviours, and is deadlock free. With Proposition 15, showing that the projections combine to admit at least the behaviour specified by the global protocol is straightforward. For the converse direction, we establish a property of the executions of the local types with respect to the global type: all the projections agree on the run taken by the overall system in the global type. We call this property *control flow agreement.* Executions that satisfy control flow agreement also satisfy protocol fidelity and are deadlock free. The formalisation and proof of this property is complicated by the fact that not all roles learn about a choice at the same time. Some roles can perform actions after the choice has been made and before they learn which branch has been taken. In the extreme case, a role may not learn at all that a choice happened. The key to control flow agreement is in the definition of the merge operator. We can simplify the reasoning to the following two points.

Roles learn choices before performing distinguishing actions. When faced with two branches with different actions, a role that is not making the choice needs to learn the branch by receiving a message. This follows from the definition of the merge operator. Let us call this message the choice message. Merging branches is only allowed as long as the actions are similar for this role. When there is a difference between two (or more) branches, an external choice is the only case that allows a role to continue on distinct branches.

Checking available messages ensures no confusion. From the possible receptions ($\mathtt{q}_i?m_i$) in an external choice, any pair of sender and message is unique among this list for the choice. This follows from two facts. First, the projection computes the available messages along the different branches of the choice. Second, merging uses that information to make sure that the choice message of one branch does not occur in another branch as a message independent of that branch's choice messages.

▶ **Example 17.** Let us use an example to illustrate why this is non-trivial. Consider:

$$\mathbf{G} := \big(\mathtt{p}\to\mathtt{q}\,{:}\,l.\,\mu t.\,\mathtt{r}\to\mathtt{p}\,{:}\,m.\,t\big) + \big(\mathtt{p}\to\mathtt{q}\,{:}\,r.\,\mu s.\,\mathtt{r}\to\mathtt{p}\,{:}\,m.\,s\big)$$

with its projections:

$$\mathbf{G}{\restriction_\mathtt{p}} = \big(\mathtt{q}!l.\,\mu t.\,\mathtt{r}?m.\,t\big) \oplus \big(\mathtt{q}!r.\,\mu s.\,\mathtt{r}?m.\,s\big) \qquad \mathbf{G}{\restriction_\mathtt{q}} = \mathtt{p}?l.\,0 \;\&\; \mathtt{p}?r.\,0 \qquad \mathbf{G}{\restriction_\mathtt{r}} = \mu t.\,\mathtt{p}!m.\,t$$

and an execution prefix w of $\{\!\{\mathsf{LAut}(\mathbf{G}{\restriction_\mathtt{p}})\}\!\}_{\mathtt{p}\in\mathcal{P}}$: $\mathtt{r}\triangleright\mathtt{p}!m.\ \mathtt{r}\triangleright\mathtt{p}!m.\ \mathtt{p}\triangleright\mathtt{q}!l.\ \mathtt{p}\triangleleft\mathtt{r}?m.\ \mathtt{r}\triangleright\mathtt{p}!m.$ For this execution prefix, we check which runs in $\mathsf{GAut}(\mathbf{G})$ each role could have pursued. In this case, $\mathtt{r}$ is not directly affected by the choice so it can proceed before the $\mathtt{p}$ has even made the choice. As the part of the protocol after the choice is a loop, we cannot bound how far some roles can proceed before the choice gets resolved. ⌟

Table 1 Projecting MSTs. For each example, we report the size as the number of nodes in the AST, the number of roles, whether it uses our extension, the time to compute the projections.

Source	Name	Size	$\|\mathcal{P}\|$	Gen. Proj. needed	Time
[52]	Instrument Contr. Prot. A	16	3	✗	0.50 ms
	Instrument Contr. Prot. B	13	3	✗	0.41 ms
	Multi Party Game	16	3	✗	0.48 ms
	OAuth2	7	3	✗	0.29 ms
	Streaming	7	4	✗	0.33 ms
[16]	Non-Compatible Merge	5	3	✓	0.22 ms
[53]	Spring-Hibernate	44	6	✓	1.97 ms
New	Group Present	43	4	✓	1.62 ms
	Late Learning	12	4	✓	0.56 ms
	Load Balancer ($n = 10$)	32	12	✓	8.18 ms
	Logging ($n = 10$)	56	13	✓	20.96 ms

▶ Remark 18. Our projection balances expressiveness with tractability: it does not unfold recursion, i.e., the merge operator never expands a term $\mu t.G$ to obtain the local type (and we only unfold once to obtain the set of available messages). Recursion variables are only handled by equality. While this restriction may seem arbitrary, unfolding can lead to comparing unbounded sequences of messages and, hence, undecidability [3] or non-effective constructions [16]. Our projection guarantees that a role is either agnostic to a choice or receives a choice message in an horizon bounded by the size of the type.

4 Evaluation

We implemented our generalised projection in a prototype tool which is publicly available [1]. The core functionality is implemented in about 800 lines of Python3 code. Our tool takes as input a global type and outputs its projections (if defined). We run our experiments on a machine with an Intel Xeon E5-2667 v2 CPU. Table 1 presents the results of our evaluation.

Our prototype successfully projects global types from the literature [52], in particular Multi-Party Game, OAuth2, Streaming, and two corrected versions of the Instrument Control Protocol. These existing examples can be projected, but do not require generalised projection.

The Need for Generalised Projection. The remaining examples exemplify when our generalised projection is needed. In fact, if some role can receive from different senders along two paths (immediately or after a sequence of same actions), its projection is only defined for the generalised projection operator. To start with, our generalised projection can project a global type presented by Castagna et al. [16, p. 19] which is not projectable with their effective projection operator (see Section 5 for more details). The Spring-Hibernate example was obtained by translating a UML sequence diagram [53] to a global type. There, `Hibernate Session` can receive from two different senders along two paths. The Group Present example is a variation of the traditional book auction example [37] and describes a protocol where friends organise a birthday present for someone; in the course of the protocol, some people can be contacted by different people. The Late Learning example models a protocol where a role submits a request and the server replies either with `reject` or `wait`, however, the last case applies to two branches where the result is served by different roles. The Load Balancer (Example 2) and Logging examples are simple versions of typical communication patterns in distributed computing. The examples are parameterised by the number of workers, respectively, back-ends that call the logging service, to evaluate the efficiency of projection. For both, we present one instance ($n = 10$) in the table. All new examples are rejected by previous approaches but shown projectable by our new projection.

Overhead. The generalised projection does not incur any overhead for global types that do not need it. Our implementation computes the sets of available messages lazily, i.e., it is only computed if our message causality analysis is needed. These sets are only needed when merging receptions from different senders. We modelled four more parameterised protocols: Mem Cache, Map Reduce, Tree Broadcast, and P2P Broadcast. We tested these examples, which do not need the generalised projection, up to size 1000 and found that our generalised projection does not add any overhead. Thus, while the message causality analysis is crucial for our generalised projection operator and hence applicability of MST verification, it does not affect its efficiency.

5 Related Work

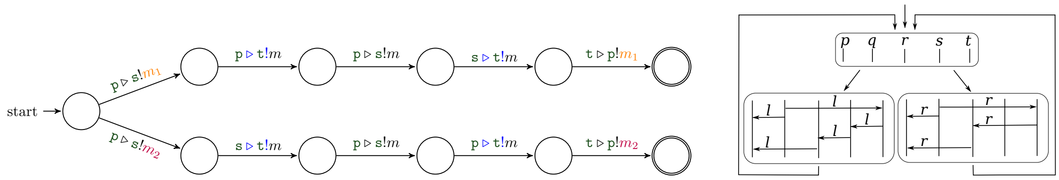
Session Types. MSTs stem from process algebra and they have been proposed for typing communication channels. The seminal work on binary session types by Honda [33] identified channel duality as a condition for safe two party communication. This work was inspired by linear logic [30] and lead to further studies on the connections between session types and linear logic [57, 13]. Moving from binary to multiparty session types, Honda et al. [36] identified consistency as the generalisation of duality for the multiparty setting. The connection between MSTs and linear logic is still ongoing [12, 14, 15]. While we focus solely on communication primitives, the theory is extended with other features such as delegation [35, 36, 18] and dependent types [54, 25, 55]. These extensions have their own intricacies and we leave incorporating such features into our generalised projection for future work.

In this paper, we use local types directly as implementations for roles for simplicity. Subtyping investigates ways to give implementation freedom while preserving the correctness properties. For further details on subtyping, we refer to work by Lange and Yoshida [43], Bravetti et al. [11], and Chen et al. [21, 20].

Generalisations of Choice in MSTs. Castagna et al. [16] consider a generalised choice similar to this work. They present a non-effective general approach for projection, relying on global information, and an algorithmic projection which is limited to local information. Our projection keeps some global information in the form of message availability and, therefore, handles a broader class of protocols. For instance, our generalised projection operator can project the following example [16, p. 19] but their algorithmic version cannot:

$$\big(\mathtt{p}\!\to\!\mathtt{r}\!:\!a.\ \mathtt{r}\!\to\!\mathtt{p}\!:\!a.\ \mathtt{p}\!\to\!\mathtt{q}\!:\!a.\ \mathtt{q}\!\to\!\mathtt{r}\!:\!b.\ 0\big) + \big(\mathtt{p}\!\to\!\mathtt{q}\!:\!a.\ \mathtt{q}\!\to\!\mathtt{r}\!:\!b.\ 0\big)$$

Hu and Yoshida [38] syntactically allow a sender to send to different recipients in global and local types as well as a receiver to receive from different senders in local types. However, their projection is only defined if a receiver receives messages from a single role. From our evaluation, all the examples that needs the generalised projection are rejected by their projection. Recently, Castellani et al. [17] investigated ways to allow local types to specify receptions from multiple senders for reversible computations but only in the synchronous



(a) Conditions for choreography automata are unsound in the asynchronous setting.

(b) Reconstructible HMSC that is not implementable.

Figure 3 These examples show that previous results for the asynchronous setting are flawed.

setting. Similarly, for synchronous communication only, Jongmans and Yoshida [40] discuss generalising choice in MSTs. Because their calculus has an explicit parallel composition, they can emulate some asynchronous communication but their channels have bag semantics instead of FIFO queues. The correctness of the projection also computes causality among messages as in our case and shares the idea of annotating local types.

Choreography Automata. Choreography automata [6] and graphical choreographies [42] model protocols as automata with transitions labelled by message exchanges, e.g., $\mathtt{p} \to \mathtt{q} : m$. Barbanera el al. [6] develop conditions for safely mergeable branches that ensure implementability on synchronous choreography automata. However, when lifting them to the asynchronous setting, they miss the subtle introduction of partial order for messages from different senders. Consider the choreography automaton in Figure 3a. It can also be represented as a global type:

$$+\begin{cases} \mathtt{p}\to\mathtt{s}:m_1.\ \mathtt{p}\to\mathtt{t}:m.\ \mathtt{p}\to\mathtt{s}:m.\ \mathtt{s}\to\mathtt{t}:m.\ \mathtt{t}\to\mathtt{p}:m_1.\ 0 \\ \mathtt{p}\to\mathtt{s}:m_2.\ \mathtt{s}\to\mathtt{t}:m.\ \mathtt{p}\to\mathtt{s}:m.\ \mathtt{p}\to\mathtt{t}:m.\ \mathtt{t}\to\mathtt{p}:m_2.\ 0 \end{cases}$$

It is *well-formed* according to their conditions. However, $\mathtt{t}$ cannot determine which branch was chosen since the messages m from $\mathtt{p}$ and $\mathtt{s}$ are not ordered when sent asynchronously. As a result, it can send m_2 in the top (resp. left) branch which is not specified as well as m_1 in the bottom (resp. right) branch.

Lange et al. [42] have shown how to obtain graphical choreographies from CSM executions. Unfortunately, they cannot fully handle unbounded FIFO channels as their method internally uses Petri nets. Still, their branching property [42, Def. 3.5] consists of similar – even though more restrictive – conditions as our MST framework: a single role chooses at each branch but roles have to learn with the first received message or do not commit any action until the branches merge back. We allow a role to learn later by recursive application of ⊓.

Implementability in Message Sequence Charts. Projection is studied in hierarchical message sequence charts (HMSCs) as *realisability*. There, variations of the problem like changing the payload of existing messages or even adding new messages in the protocol are also considered [49, 29]. Here, we focus on implementability without altering the protocol. HMSCs are a more general model than MSTs and, unsurprisingly, realisability is undecidable [28, 3]. Thus, restricted models have been studied. Boundedness [3] is one such example: checking safe realisability for bounded HMSCs is EXPSPACE-complete [45]. Boundedness is a very strong restriction. Weaker restrictions, as in MSTs, center on choice. As we explain below, these restrictions are either flawed, overly restrictive, or not effectively checkable.

The first definition of (non-)local choice for HMSCs by Ben-Abdallah and Leue [7] suffers from severely restrictive assumptions and only yields finite-state systems. Given an HMSC specification, research on *implied scenarios*, e.g. by Muccini et al. [50], investigates whether there are behaviours which, due to the asynchronous nature of communication, every implementation must allow in addition to the specified ones. In our setting, an implementable protocol specification must not have any implied scenarios. Mooij et al. [48] point out several contradictions of the observations on implied scenarios and non-local choice. Hence, they propose more variants of non-local choices but allow implied scenarios. In our setting, this corresponds to allowing roles to follow different branches.

Similar to allowing implied scenarios of specifications, Hélouët [31] pointed out that non-local choice has been frequently misunderstood as it actually does not ensure implementability but less ambiguity. Hélouët and Jard proposed the notion of reconstructibility [32] for a quite restrictive setting: first, messages need to be unique in the protocol specification and, second, each node in an HMSC is implicitly terminal. Unfortunately, we show their results are flawed. Consider the HMSC in Figure 3b. (For simplicity, we use the same message identifier in each branch but one can easily index them for uniqueness.) The same protocol can be represented by the following global type:

$$\mu t.\ + \begin{cases} \mathtt{q}\!\rightarrow\!\mathtt{t}\!:\!l.\ \mathtt{q}\!\rightarrow\!\mathtt{p}\!:\!l.\ \mathtt{t}\!\rightarrow\!\mathtt{s}\!:\!l.\ \mathtt{s}\!\rightarrow\!\mathtt{r}\!:\!l.\ \mathtt{r}\!\rightarrow\!\mathtt{p}\!:\!l.\ t \\ \mathtt{q}\!\rightarrow\!\mathtt{t}\!:\!r.\ \mathtt{q}\!\rightarrow\!\mathtt{p}\!:\!r.\ \mathtt{t}\!\rightarrow\!\mathtt{r}\!:\!r.\ \mathtt{r}\!\rightarrow\!\mathtt{p}\!:\!r.\ t \end{cases}$$

Because their notion of reconstructibility [32, Def. 12] only considers loop-free paths, they report that the HMSC is reconstructible. However, the HMSC is not implementable. Suppose that $\mathtt{q}$ first chooses to take the top (resp. left) and then the bottom (resp. right) branch. The message l from $\mathtt{s}$ to $\mathtt{r}$ can be delayed until after $\mathtt{r}$ received r from $\mathtt{t}$. Therefore, $\mathtt{r}$ will first send r to $\mathtt{p}$ and then l which contradicts with the order of branches taken. This counterexample contradicts their result [32, Thm. 15] and shows that reconstructibility is not sufficient for implementability.

Dan et al. [22], improving Baker et al. [5], provide a definition that ensures implementability. They provide a definition which is based on projected words of the HMSC in contrast to the choices. It is unknown how to check their condition for HMSCs.

CSMs and MSTs. The connection between MSTs, CSMs, and automata [16, 24] came shortly after the introduction of MSTs. Denielou and Yoshida [24] use CSMs but they preserve the restrictions on choice from MSTs. It is well-known that CSMs are Turing-powerful [10]. Decidable instances of CSM verification can be obtained by restricting the communication topology [51, 56] or by altering the semantics of communication, e.g. by making channels lossy [2], half-duplex [19], or input-bounded [9]. Lange and Yoshida [44] proposed additional notions that resemble ideas from MSTs.

6 Conclusion

We have presented a generalised projection operator for asynchronous MSTs. The key challenge lies in the generalisation of the external choice to allow roles to receive from more than one sender. We provide a new projectability check and a soundness theorem that shows projectability implies implementability. The key to our results is a message causality analysis and an automata-theoretic soundness proof. With a prototype implementation, we have demonstrated that our MST framework can project examples from the literature as well as new examples, including typical communication patterns in distributed computing, which were not projectable by previous projection operators.

References

1 Prototype Implementation of Generalised Projection for Multiparty Session Types. URL: `https://gitlab.mpi-sws.org/fstutz/async-mpst-gen-choice/`.

2 Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson. On-the-fly analysis of systems with unbounded, lossy FIFO channels. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer Aided Verification, 10th International Conference, CAV'98, Vancouver, BC, Canada, June 28 – July 2, 1998, Proceedings*, volume 1427 of *Lecture Notes in Computer Science*, pages 305–318. Springer, 1998. `doi:10.1007/BFb0028754`.

3 Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Realizability and verification of MSC graphs. *Theor. Comput. Sci.*, 331(1):97–114, 2005. `doi:10.1016/j.tcs.2004.09.034`.

4 Davide Ancona, Viviana Bono, Mario Bravetti, Joana Campos, Giuseppe Castagna, Pierre-Malo Deniélou, Simon J. Gay, Nils Gesbert, Elena Giachino, Raymond Hu, Einar Broch Johnsen, Francisco Martins, Viviana Mascardi, Fabrizio Montesi, Rumyana Neykova, Nicholas Ng, Luca Padovani, Vasco T. Vasconcelos, and Nobuko Yoshida. Behavioral types in programming languages. *Found. Trends Program. Lang.*, 3(2-3):95–230, 2016. `doi:10.1561/2500000031`.

5 Paul Baker, Paul Bristow, Clive Jervis, David J. King, Robert Thomson, Bill Mitchell, and Simon Burton. Detecting and resolving semantic pathologies in UML sequence diagrams. In Michel Wermelinger and Harald C. Gall, editors, *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2005, Lisbon, Portugal, September 5-9, 2005*, pages 50–59. ACM, 2005. `doi:10.1145/1081706.1081716`.

6 Franco Barbanera, Ivan Lanese, and Emilio Tuosto. Choreography automata. In Simon Bliudze and Laura Bocchi, editors, *Coordination Models and Languages – 22nd IFIP WG 6.1 International Conference, COORDINATION 2020, Held as Part of the 15th International Federated Conference on Distributed Computing Techniques, DisCoTec 2020, Valletta, Malta, June 15-19, 2020, Proceedings*, volume 12134 of *Lecture Notes in Computer Science*, pages 86–106. Springer, 2020. `doi:10.1007/978-3-030-50029-0_6`.

7 Hanêne Ben-Abdallah and Stefan Leue. Syntactic detection of process divergence and non-local choice inmessage sequence charts. In Ed Brinksma, editor, *Tools and Algorithms for Construction and Analysis of Systems, Third International Workshop, TACAS '97, Enschede, The Netherlands, April 2-4, 1997, Proceedings*, volume 1217 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 1997. `doi:10.1007/BFb0035393`.

8 Laura Bocchi, Maurizio Murgia, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Asynchronous timed session types – from duality to time-sensitive processes. In Luís Caires, editor, *Programming Languages and Systems – 28th European Symposium on Programming, ESOP 2019, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2019, Prague, Czech Republic, April 6-11, 2019, Proceedings*, volume 11423 of *Lecture Notes in Computer Science*, pages 583–610. Springer, 2019. `doi:10.1007/978-3-030-17184-1_21`.

9 Benedikt Bollig, Alain Finkel, and Amrita Suresh. Bounded reachability problems are decidable in FIFO machines. In Igor Konnov and Laura Kovács, editors, *31st International Conference on Concurrency Theory, CONCUR 2020, September 1-4, 2020, Vienna, Austria (Virtual Conference)*, volume 171 of *LIPIcs*, pages 49:1–49:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPIcs.CONCUR.2020.49`.

10 Daniel Brand and Pitro Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983. `doi:10.1145/322374.322380`.

11 Mario Bravetti, Marco Carbone, and Gianluigi Zavattaro. On the boundary between decidability and undecidability of asynchronous session subtyping. *Theor. Comput. Sci.*, 722:19–51, 2018. `doi:10.1016/j.tcs.2018.02.010`.

12 Luís Caires and Jorge A. Pérez. Multiparty session types within a canonical binary theory, and beyond. In Elvira Albert and Ivan Lanese, editors, *Formal Techniques for Distributed Objects, Components, and Systems – 36th IFIP WG 6.1 International Conference, FORTE 2016, Held as Part of the 11th International Federated Conference on Distributed Computing Techniques, DisCoTec 2016, Heraklion, Crete, Greece, June 6-9, 2016, Proceedings*, volume 9688 of *Lecture Notes in Computer Science*, pages 74–95. Springer, 2016. doi:10.1007/978-3-319-39570-8_6.

13 Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Math. Struct. Comput. Sci.*, 26(3):367–423, 2016. doi:10.1017/S0960129514000218.

14 Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schürmann, and Philip Wadler. Coherence generalises duality: A logical explanation of multiparty session types. In Josée Desharnais and Radha Jagadeesan, editors, *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, volume 59 of *LIPIcs*, pages 33:1–33:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.33.

15 Marco Carbone, Fabrizio Montesi, Carsten Schürmann, and Nobuko Yoshida. Multiparty session types as coherence proofs. *Acta Informatica*, 54(3):243–269, 2017. doi:10.1007/s00236-016-0285-y.

16 Giuseppe Castagna, Mariangiola Dezani-Ciancaglini, and Luca Padovani. On global types and multi-party session. *Log. Methods Comput. Sci.*, 8(1), 2012. doi:10.2168/LMCS-8(1:24)2012.

17 Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Paola Giannini. Reversible sessions with flexible choices. *Acta Informatica*, 56(7-8):553–583, 2019. doi:10.1007/s00236-019-00332-y.

18 Ilaria Castellani, Mariangiola Dezani-Ciancaglini, Paola Giannini, and Ross Horne. Global types with internal delegation. *Theor. Comput. Sci.*, 807:128–153, 2020. doi:10.1016/j.tcs.2019.09.027.

19 Gérard Cécé and Alain Finkel. Verification of programs with half-duplex communication. *Inf. Comput.*, 202(2):166–190, 2005. doi:10.1016/j.ic.2005.05.006.

20 Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. On the preciseness of subtyping in session types. *Log. Methods Comput. Sci.*, 13(2), 2017. doi:10.23638/LMCS-13(2:12)2017.

21 Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. On the preciseness of subtyping in session types. In Olaf Chitil, Andy King, and Olivier Danvy, editors, *Proceedings of the 16th International Symposium on Principles and Practice of Declarative Programming, Kent, Canterbury, United Kingdom, September 8-10, 2014*, pages 135–146. ACM, 2014. doi:10.1145/2643135.2643138.

22 Haitao Dan, Robert M. Hierons, and Steve Counsell. Non-local choice and implied scenarios. In José Luiz Fiadeiro, Stefania Gnesi, and Andrea Maggiolo-Schettini, editors, *8th IEEE International Conference on Software Engineering and Formal Methods, SEFM 2010, Pisa, Italy, 13-18 September 2010*, pages 53–62. IEEE Computer Society, 2010. doi:10.1109/SEFM.2010.14.

23 Ankush Das, Stephanie Balzer, Jan Hoffmann, and Frank Pfenning. Resource-aware session types for digital contracts. *CoRR*, abs/1902.06056, 2019. arXiv:1902.06056.

24 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty session types meet communicating automata. In Helmut Seidl, editor, *Programming Languages and Systems – 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 – April 1, 2012. Proceedings*, volume 7211 of *Lecture Notes in Computer Science*, pages 194–213. Springer, 2012. doi:10.1007/978-3-642-28869-2_10.

25 Pierre-Malo Deniélou, Nobuko Yoshida, Andi Bejleri, and Raymond Hu. Parameterised multiparty session types. *Log. Methods Comput. Sci.*, 8(4), 2012. doi:10.2168/LMCS-8(4:6)2012.

26 Manuel Fähndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen C. Hunt, James R. Larus, and Steven Levi. Language support for fast and reliable message-based communication in singularity OS. In Yolande Berbers and Willy Zwaenepoel, editors, *Proceedings of the 2006 EuroSys Conference, Leuven, Belgium, April 18-21, 2006*, pages 177–190. ACM, 2006. doi:10.1145/1217935.1217953.

27 Paul Gastin. Infinite traces. In Irène Guessarian, editor, *Semantics of Systems of Concurrent Processes, LITP Spring School on Theoretical Computer Science, La Roche Posay, France, April 23-27, 1990, Proceedings*, volume 469 of *Lecture Notes in Computer Science*, pages 277–308. Springer, 1990. doi:10.1007/3-540-53479-2_12.

28 Blaise Genest, Anca Muscholl, and Doron A. Peled. Message sequence charts. In Jörg Desel, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Lectures on Concurrency and Petri Nets, Advances in Petri Nets [This tutorial volume originates from the 4th Advanced Course on Petri Nets, ACPN 2003, held in Eichstätt, Germany in September 2003. In addition to lectures given at ACPN 2003, additional chapters have been commissioned]*, volume 3098 of *Lecture Notes in Computer Science*, pages 537–558. Springer, 2003. doi:10.1007/978-3-540-27755-2_15.

29 Blaise Genest, Anca Muscholl, Helmut Seidl, and Marc Zeitoun. Infinite-state high-level mscs: Model-checking and realizability. *J. Comput. Syst. Sci.*, 72(4):617–647, 2006. doi:10.1016/j.jcss.2005.09.007.

30 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.

31 Loïc Hélouët. Some pathological message sequence charts, and how to detect them. In Rick Reed and Jeanne Reed, editors, *SDL 2001: Meeting UML, 10th International SDL Forum Copenhagen, Denmark, June 27-29, 2001, Proceedings*, volume 2078 of *Lecture Notes in Computer Science*, pages 348–364. Springer, 2001. doi:10.1007/3-540-48213-X_22.

32 Loïc Hélouët and Claude Jard. Conditions for synthesis of communicating automata from HMSCs. In *In 5th International Workshop on Formal Methods for Industrial Critical Systems (FMICS)*, 2000.

33 Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993. doi:10.1007/3-540-57208-2_35.

34 Kohei Honda, Eduardo R. B. Marques, Francisco Martins, Nicholas Ng, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Verification of MPI programs using session types. In Jesper Larsson Träff, Siegfried Benkner, and Jack J. Dongarra, editors, *Recent Advances in the Message Passing Interface – 19th European MPI Users' Group Meeting, EuroMPI 2012, Vienna, Austria, September 23-26, 2012. Proceedings*, volume 7490 of *Lecture Notes in Computer Science*, pages 291–293. Springer, 2012. doi:10.1007/978-3-642-33518-1_37.

35 Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In Chris Hankin, editor, *Programming Languages and Systems – ESOP'98, 7th European Symposium on Programming, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'98, Lisbon, Portugal, March 28 – April 4, 1998, Proceedings*, volume 1381 of *Lecture Notes in Computer Science*, pages 122–138. Springer, 1998. doi:10.1007/BFb0053567.

36 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 273–284. ACM, 2008. doi:10.1145/1328438.1328472.

37 Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. *J. ACM*, 63(1):9:1–9:67, 2016. doi:10.1145/2827695.

38 Raymond Hu and Nobuko Yoshida. Explicit connection actions in multiparty session types. In Marieke Huisman and Julia Rubin, editors, *Fundamental Approaches to Software Engineering – 20th International Conference, FASE 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10202 of *Lecture Notes in Computer Science*, pages 116–133. Springer, 2017. doi:10.1007/978-3-662-54494-5_7.

39 Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016. `doi:10.1145/2873052`.

40 Sung-Shik Jongmans and Nobuko Yoshida. Exploring type-level bisimilarity towards more expressive multiparty session types. In Peter Müller, editor, *Programming Languages and Systems – 29th European Symposium on Programming, ESOP 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12075 of *Lecture Notes in Computer Science*, pages 251–279. Springer, 2020. `doi:10.1007/978-3-030-44914-8_10`.

41 Dimitrios Kouzapas, Ramunas Gutkovas, A. Laura Voinea, and Simon J. Gay. A session type system for asynchronous unreliable broadcast communication. *CoRR*, abs/1902.01353, 2019. `arXiv:1902.01353`.

42 Julien Lange, Emilio Tuosto, and Nobuko Yoshida. From communicating machines to graphical choreographies. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 221–232. ACM, 2015. `doi:10.1145/2676726.2676964`.

43 Julien Lange and Nobuko Yoshida. On the undecidability of asynchronous session subtyping. In Javier Esparza and Andrzej S. Murawski, editors, *Foundations of Software Science and Computation Structures – 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*, volume 10203 of *Lecture Notes in Computer Science*, pages 441–457, 2017. `doi:10.1007/978-3-662-54458-7_26`.

44 Julien Lange and Nobuko Yoshida. Verifying asynchronous interactions via communicating session automata. In Isil Dillig and Serdar Tasiran, editors, *Computer Aided Verification – 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part I*, volume 11561 of *Lecture Notes in Computer Science*, pages 97–117. Springer, 2019. `doi:10.1007/978-3-030-25540-4_6`.

45 Markus Lohrey. Realizability of high-level message sequence charts: closing the gaps. *Theor. Comput. Sci.*, 309(1-3):529–554, 2003. `doi:10.1016/j.tcs.2003.08.002`.

46 Rupak Majumdar, Madhavan Mukund, Felix Stutz, and Damien Zufferey. Generalising projection in asynchronous multiparty session types. *CoRR*, abs/2107.03984, 2021. `arXiv:2107.03984`.

47 Rupak Majumdar, Marcus Pirron, Nobuko Yoshida, and Damien Zufferey. Motion session types for robotic interactions (brave new idea paper). In Alastair F. Donaldson, editor, *33rd European Conference on Object-Oriented Programming, ECOOP 2019, July 15-19, 2019, London, United Kingdom*, volume 134 of *LIPIcs*, pages 28:1–28:27. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. `doi:10.4230/LIPIcs.ECOOP.2019.28`.

48 Arjan J. Mooij, Nicolae Goga, and Judi Romijn. Non-local choice and beyond: Intricacies of MSC choice nodes. In Maura Cerioli, editor, *Fundamental Approaches to Software Engineering, 8th International Conference, FASE 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3442 of *Lecture Notes in Computer Science*, pages 273–288. Springer, 2005. `doi:10.1007/978-3-540-31984-9_21`.

49 Rémi Morin. Recognizable sets of message sequence charts. In Helmut Alt and Afonso Ferreira, editors, *STACS 2002, 19th Annual Symposium on Theoretical Aspects of Computer Science, Antibes – Juan les Pins, France, March 14-16, 2002, Proceedings*, volume 2285 of *Lecture Notes in Computer Science*, pages 523–534. Springer, 2002. `doi:10.1007/3-540-45841-7_43`.

50 Henry Muccini. Detecting implied scenarios analyzing non-local branching choices. In Mauro Pezzè, editor, *Fundamental Approaches to Software Engineering, 6th International Conference, FASE 2003, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2003, Warsaw, Poland, April 7-11, 2003, Proceedings*, volume 2621 of *Lecture Notes in Computer Science*, pages 372–386. Springer, 2003. `doi:10.1007/3-540-36578-8_26`.

51 Wuxu Peng and S. Purushothaman. Analysis of a class of communicating finite state machines. *Acta Informatica*, 29(6/7):499–522, 1992. doi:10.1007/BF01185558.

52 Alceste Scalas and Nobuko Yoshida. Less is more: multiparty session types revisited. *Proc. ACM Program. Lang.*, 3(POPL):30:1–30:29, 2019. doi:10.1145/3290343.

53 Spring and Hibernate Transaction in Java. URL: https://www.uml-diagrams.org/examples/spring-hibernate-transaction-sequence-diagram-example.html.

54 Bernardo Toninho, Luís Caires, and Frank Pfenning. Dependent session types via intuitionistic linear type theory. In Peter Schneider-Kamp and Michael Hanus, editors, *Proceedings of the 13th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming, July 20-22, 2011, Odense, Denmark*, pages 161–172. ACM, 2011. doi:10.1145/2003476.2003499.

55 Bernardo Toninho and Nobuko Yoshida. Depending on session-typed processes. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures – 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 128–145. Springer, 2018. doi:10.1007/978-3-319-89366-2_7.

56 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Context-bounded analysis of concurrent queue systems. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2008. doi:10.1007/978-3-540-78800-3_21.

57 Philip Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014. doi:10.1017/S095679681400001X.

58 Nobuko Yoshida, Raymond Hu, Rumyana Neykova, and Nicholas Ng. The scribble protocol language. In Martín Abadi and Alberto Lluch-Lafuente, editors, *Trustworthy Global Computing – 8th International Symposium, TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers*, volume 8358 of *Lecture Notes in Computer Science*, pages 22–41. Springer, 2013. doi:10.1007/978-3-319-05119-2_3.

A Communicating State Machines

A *communicating state machine* (CSM) $\mathcal{A} = \{\!\{A_{\mathsf{p}}\}\!\}_{\mathsf{p}\in\mathcal{P}}$ over $\mathcal{P}$ and $\mathcal{V}$ consists of a state machine A_{p} over Σ_{p} for each $\mathsf{p} \in \mathcal{P}$. A state machine for p will be denoted by $(Q_{\mathsf{p}}, \Sigma_{\mathsf{p}}, \delta_{\mathsf{p}}, q_{0,\mathsf{p}}, F_{\mathsf{p}})$. If a state q has multiple outgoing transitions, all labelled with send actions, then q is called an *internal choice* state. If all the outgoing transitions are labelled with receive actions, q is called an *external choice* state. Otherwise, q is a *mixed choice* state. In this paper, we only consider state machines without mixed choice states.

Intuitively, a CSM represents a set of state machines, one for each role in $\mathcal{P}$, interacting via message sends and receipts. We assume that each pair of roles $\mathsf{p}, \mathsf{q} \in \mathcal{P}$, $\mathsf{p} \neq \mathsf{q}$, is connected by a *message channel*. A transition $q_{\mathsf{p}} \xrightarrow{\mathsf{p}\triangleright\mathsf{q}!m} q'_{\mathsf{p}}$ in the state machine of p specifies that, when p is in the state q_{p}, it sends a message m to q, and updates its local state to q'_{p}. The message m is appended to the channel $\langle \mathsf{p}, \mathsf{q}\rangle$. Similarly, a transition $q_{\mathsf{q}} \xrightarrow{\mathsf{q}\triangleleft\mathsf{p}?m} q'_{\mathsf{q}}$ in the state machine of q specifies that q in state q_{q} can retrieve the message m from the head of the channel $\langle \mathsf{p}, \mathsf{q}\rangle$ and update its local state to q'_{q}.

Let $\mathsf{Chan} = \{\langle \mathsf{p}, \mathsf{q}\rangle \mid \mathsf{p}, \mathsf{q} \in \mathcal{P}, \mathsf{p} \neq \mathsf{q}\}$ denote the set of channels. The set of global states of the CSM is given by $\prod_{\mathsf{p}\in\mathcal{P}} Q_{\mathsf{p}}$. For a global state q, we write q_{p} for the state of p in q. A *configuration* of $\mathcal{A}$ is a pair (q, ξ), where q is a global state and $\xi : \mathsf{Chan} \to \mathcal{V}^*$ maps each channel to the queue of messages currently in the channel. The initial configuration is (q_0, ξ_ε), where $q_{0,\mathtt{p}}$ is the initial state of $A_\mathtt{p}$ for each $\mathtt{p} \in \mathcal{P}$ and ξ_ε maps each channel to ε. A configuration (q, ξ) is *final* iff $q_\mathtt{p}$ is final for every $\mathtt{p}$ and $\xi = \xi_\varepsilon$.

In a global move of a CSM, a single role executes a local transition to change its state, while all other roles remain stationary. For a send or a receive action, the corresponding channel is updated, while all other channels remain unchanged. Formally, the global transition relation $\rightarrow$ on configurations is defined as follows:

- $(q, \xi) \xrightarrow{\mathtt{p}\triangleright\mathtt{q}!m} (q', \xi')$ if $(q_\mathtt{p}, \mathtt{p}\triangleright\mathtt{q}!m, q'_\mathtt{p}) \in \delta_\mathtt{p}$, $q_\mathtt{r} = q'_\mathtt{r}$ for every $\mathtt{r} \neq \mathtt{p}$, $\xi'(\langle \mathtt{p}, \mathtt{q}\rangle) = \xi(\langle \mathtt{p}, \mathtt{q}\rangle)\cdot m$ and $\xi'(c) = \xi(c)$ for every other channel $c \in \mathsf{Chan}$.
- $(q, \xi) \xrightarrow{\mathtt{q}\triangleleft\mathtt{p}?m} (q', \xi')$ if $(q_\mathtt{q}, \mathtt{q}\triangleleft\mathtt{p}?m, q'_\mathtt{q}) \in \delta_\mathtt{q}$, $q_\mathtt{r} = q'_\mathtt{r}$ for every $\mathtt{r} \neq \mathtt{q}$, $\xi(\langle \mathtt{p}, \mathtt{q}\rangle) = m\cdot\xi'(\langle \mathtt{p}, \mathtt{q}\rangle)$ and $\xi'(c) = \xi(c)$ for every other channel $c \in \mathsf{Chan}$.
- $(q, \xi) \xrightarrow{\tau} (q', \xi)$ if $(q_\mathtt{p}, \varepsilon, q'_\mathtt{p}) \in \delta_\mathtt{p}$ for some role $\mathtt{p}$, and $q_\mathtt{q} = q'_\mathtt{q}$ for every role $\mathtt{q} \neq \mathtt{p}$.

A run of the CSM is a finite or infinite sequence: $(q_0, \xi_0) \xrightarrow{x_0} (q_1, \xi_1) \xrightarrow{x_1} \ldots$, such that (q_0, ξ_0) is the initial configuration and for each $i \geq 0$, we have $(q_i, \xi_i) \xrightarrow{x_i} (q_{i+1}, \xi_{i+1})$. The *trace* of the run is the word $x_0 x_1 \ldots \in \Sigma^\infty$. We also call $x_0 x_1 \ldots$ an *execution prefix*. A run is maximal if it is infinite or if it is finite and ends in a final configuration. A trace is maximal if it is the trace of a maximal run. The language $\mathcal{L}(\mathcal{A})$ of the CSM $\mathcal{A}$ is the set of maximal traces. A CSM is *deadlock free* if every finite run can be extended to a maximal run.

The following lemma summarises some properties of execution prefixes of CSMs. The proofs are by induction on the length of the run.

▶ **Lemma 19.** *Let $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ be a CSM. For any run $(q_0, \xi_0) \xrightarrow{x_0} \cdots \xrightarrow{x_n} (q, \xi)$ with trace $w = x_0 \ldots x_n$, it holds that (1) $\xi(\langle \mathtt{p}, \mathtt{q}\rangle) = u$ where $\mathcal{V}(w\Downarrow_{\mathtt{p}\triangleright\mathtt{q}!\_}) = \mathcal{V}(w\Downarrow_{\mathtt{q}\triangleleft\mathtt{p}?\_}).u$ for every pair of roles $\mathtt{p}, \mathtt{q} \in \mathcal{P}$ and (2) w is channel-compliant. Maximal traces of $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ are channel-compliant and complete.*

Proof. We prove the claims by induction on an execution prefix w. The base case where $w = \varepsilon$ is trivial. For the induction step, we consider wx with the following run in $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$: $(q_0, \xi_0) \xrightarrow{w} (q, \xi) \xrightarrow{x} (q', \xi')$. The induction hypothesis holds for w and (q, ξ) and we prove the claims for (q', ξ') and wx. We do a case analysis on x. If $x = \tau$, the claim trivially follows.

Let $x = \mathtt{q} \triangleleft \mathtt{p}?m$. From the induction hypothesis, we know that $\xi(\langle \mathtt{p}, \mathtt{q}\rangle) = u$ where $\mathcal{V}(w\Downarrow_{\mathtt{p}\triangleright\mathtt{q}!\_}) = \mathcal{V}(w\Downarrow_{\mathtt{q}\triangleleft\mathtt{p}?\_}).u$. Since x is a possible transition, we know that $u = m.u'$ for some u' and $\xi'(\langle \mathtt{p}, \mathtt{q}\rangle) = u'$. By definition, it holds that $\mathcal{V}(w\Downarrow_{\mathtt{q}\triangleleft\mathtt{p}?\_}).m.u' = \mathcal{V}((wx)\Downarrow_{\mathtt{q}\triangleleft\mathtt{p}?\_}).u'$. For all other pairs of roles, the induction hypothesis applies since the above projections do not change. Hence, wx is channel-compliant.

Let $x = \mathtt{p} \triangleright \mathtt{q}!m$. From the induction hypothesis, we know that $\xi(\langle \mathtt{p}, \mathtt{q}\rangle) = u$ where $\mathcal{V}(w\Downarrow_{\mathtt{p}\triangleright\mathtt{q}!\_}) = \mathcal{V}(w\Downarrow_{\mathtt{q}\triangleleft\mathtt{p}?\_}).u$. Since x is a possible transition, we know that $\xi'(\langle \mathtt{p}, \mathtt{q}\rangle) = u.m$. By definition and induction hypothesis, we have: $\mathcal{V}((wx)\Downarrow_{\mathtt{p}\triangleright\mathtt{q}!\_}) = \mathcal{V}(w\Downarrow_{\mathtt{p}\triangleright\mathtt{q}!\_}).m = \mathcal{V}(w\Downarrow_{\mathtt{q}\triangleleft\mathtt{p}?\_}).u.m$. For all other combinations of roles, the induction hypothesis applies since the above projections do not change. Hence, wx is channel-compliant.

From (1) and (2), it follows directly that maximal traces of $\{\!\{A_\mathtt{p}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ are channel-compliant and complete. ◀

B Properties of $\mathcal{C}^\sim$

▶ **Lemma 20.** *Let $L \subseteq \Sigma_\mathtt{p}^\infty$. Then, $L = \mathcal{C}^\sim(L)$.*

Proof. For any $w \in \Sigma_\mathtt{p}^\infty$, none of the rules of $\sim_1$ applies to w, and we have that $w \sim w'$ iff $w = w'$. Thus, $L = \mathcal{C}^\sim(L)$ for any language $L \subseteq \Sigma_\mathtt{p}^\infty$. ◀

▶ **Lemma 21.** *Let $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ be a CSM. Then, for every finite w with a run in $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ and every $w' \sim w$, w' has a run that ends with the same configuration. The language $\mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}})$ is closed under $\sim$: $\mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}) = \mathcal{C}^{\sim}(\mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}))$.*

Proof. Let w be a finite word with a run in $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ and $w' \sim w$. By definition, $w' \sim_n w$ for some n. We prove that w' has a run that ends in the same configuration by induction on n. The base case for $n = 0$ is trivial. For the induction step, we assume that the claim holds for n and prove it for $n+1$. Suppose that $w \sim_{n+1} w'$. Then, there is w'' such that $w' \sim_1 w''$ and $w'' \sim_n w$. By assumption, we know that $w' = u'u_1u_2u''$ and $w'' = u'u_2u_1u''$ for some $u', u'' \in \Sigma^*$, $u_1, u_2 \in \Sigma$. By induction hypothesis, we know that $w'' \in \mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}})$ so there is run for w'' in $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$. Let us investigate the run at u_1 and u_2: $\cdots (q_1, \xi_1) \xrightarrow{u_1} (q_2, \xi_2) \xrightarrow{u_2} (q_3, \xi_3) \cdots$. It suffices to show that $\cdots (q_1, \xi_1) \xrightarrow{u_2} (q'_2, \xi'_2) \xrightarrow{u_1} (q_3, \xi_3) \cdots$ is possible in $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ for some configuration (q'_2, ξ'_2). We do a case analysis on the rule that was applied for $w' \sim_1 w''$.

- $u_1 = \mathtt{p} \triangleright \mathtt{q}!m$, $u_2 = \mathtt{r} \triangleright \mathtt{s}!m'$, and $\mathtt{p} \neq \mathtt{r}$:
  We define q'_2 such that $q'_{2,\mathtt{p}} = q_{1,\mathtt{p}}$, $q'_{2,\mathtt{r}} = q_{3,\mathtt{r}}$, and $q'_{2,\mathtt{t}} = q_{3,\mathtt{t}}$ for all $\mathtt{t} \in \mathcal{P}$ with $\mathtt{t} \neq \mathtt{p}$ and $\mathtt{t} \neq \mathtt{r}$. Both transitions are feasible in $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ because both $\mathtt{p}$ and $\mathtt{r}$ are different and send a message to different channels. They can do this independently from each other.
- $u_1 = \mathtt{q} \triangleleft \mathtt{p}?m$, $u_2 = \mathtt{s} \triangleleft \mathtt{r}?m'$, and $\mathtt{q} \neq \mathtt{s}$:
  We define q'_2 such that $q'_{2,\mathtt{q}} = q_{1,\mathtt{q}}$, $q'_{2,\mathtt{s}} = q_{3,\mathtt{s}}$, and $q'_{2,\mathtt{t}} = q_{3,\mathtt{t}}$ for all $\mathtt{t} \in \mathcal{P}$ with $\mathtt{t} \neq \mathtt{p}$ and $\mathtt{t} \neq \mathtt{r}$. Both transitions are feasible in $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ because both $\mathtt{q}$ and $\mathtt{s}$ are different and receive a message from a different channel. They can do this independently from each other.
- $u_1 = \mathtt{p} \triangleright \mathtt{q}!m$, $u_2 = \mathtt{s} \triangleleft \mathtt{r}?m'$, and and $\mathtt{p} \neq \mathtt{s} \wedge (\mathtt{p} \neq \mathtt{r} \vee \mathtt{q} \neq \mathtt{s})$.
  We define q'_2 such that $q'_{2,\mathtt{p}} = q_{1,\mathtt{p}}$, $q'_{2,\mathtt{s}} = q_{3,\mathtt{s}}$, and $q'_{2,\mathtt{t}} = q_{3,\mathtt{t}}$ for all $\mathtt{t} \in \mathcal{P}$ with $\mathtt{t} \neq \mathtt{p}$ and $\mathtt{t} \neq \mathtt{r}$. Let us do a case split according to the side conditions. First, let $\mathtt{p} \neq \mathtt{s}$ and $\mathtt{p} \neq \mathtt{r}$. The channels of u_1 and u_2 are different and $\mathtt{p}$ and $\mathtt{s}$ are different, so both transitions are feasible in $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$.
  Second, let $\mathtt{p} \neq \mathtt{s}$ and $\mathtt{q} \neq \mathtt{s}$. The channels of u_1 and u_2 are different and $\mathtt{q}$ and $\mathtt{s}$ are different, so both transitions are feasible in $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$.
- $u_1 = \mathtt{p} \triangleright \mathtt{q}!m$, $u_2 = \mathtt{q} \triangleleft \mathtt{p}?m'$, and $|u' \Downarrow_{\mathtt{p}\triangleright\mathtt{q}!\_}| > |u' \Downarrow_{\mathtt{q}\triangleleft\mathtt{p}?\_}|$:
  We define q'_2 such that $q'_{2,\mathtt{p}} = q_{1,\mathtt{p}}$, $q'_{2,\mathtt{q}} = q_{3,\mathtt{q}}$, and $q'_{2,\mathtt{t}} = q_{3,\mathtt{t}}$ for all $\mathtt{t} \neq \mathtt{p}$ and $\mathtt{t} \neq \mathtt{q}$. In this case, the channel of u_1 and u_2 is the same but the side condition ensures that u_2 actually has a different message read since the channel $\xi_1(\langle \mathtt{p}, \mathtt{q}\rangle)$ is not empty by Lemma 19 and, hence, both transitions can act independently and lead to the same configuration.

This proves that w' has a run in $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ that ends in the same configuration which concludes the proof of the first claim.

For the second claim, we know that $\mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}) \subseteq \mathcal{C}^{\sim}(\mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}))$ by definition. Hence, it suffices to show that $\mathcal{C}^{\sim}(\mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}})) \subseteq \mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}})$.

We show the claim for finite traces first:

$$\mathcal{C}^{\sim}(\mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}})) \cap \Sigma^* \subseteq \mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}) \cap \Sigma^*.$$

Let $w' \in \mathcal{C}^{\sim}(\mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}})) \cap \Sigma^*$. There is $w \in \mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}) \cap \Sigma^*$ such that $w \sim w'$. By definition, w has a run in $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ which ends in a final configuration. From the first claim, we know that w' also has a run that ends in the same configuration which is final. Therefore, $w \in \mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}) \cap \Sigma^*$. Hence, the claim holds for finite traces.

It remains to show the claim for infinite traces. To this end, we show that for every execution prefix w of $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ such that $w \sim u$ for $u \in \text{pref}(\mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}))$ and any continuation x of w, i.e., wx is an execution prefix of $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$, it holds that $wx \sim ux$ and $ux \in \text{pref}(\mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}))$ (□). We know that $w \sim_n u$ for some n by definition, so $wx \sim_n ux$ since we can mimic the same n swaps when extending both w and u by x. From the first claim, we know that $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ is in the same configuration (q, ξ) after processing w and u. Therefore, ux is an execution prefix of $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$ because wx is which yields (□).

We show that

$$\mathcal{C}^{\sim}(\mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}) \cap \Sigma^{\omega} \subseteq \mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}) \cap \Sigma^{\omega}.$$

Let $w \in \mathcal{C}^{\sim}(\mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}) \cap \Sigma^{\omega}$. We show that w has an infinite run in $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$.

Consider a tree $\mathcal{T}$ where each node corresponds to a run ρ on some finite prefix $w' \leq w$ in $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$. The root is labelled by the empty run. The children of a node ρ are runs that extend ρ by a single transition – these exist by (□). Since our CSM, derived from a global type, is built from a finite number of finite state machines, $\mathcal{T}$ is finitely branching. By König's Lemma, there is an infinite path in $\mathcal{T}$ that corresponds to an infinite run for w in $\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}$, so $w \in \mathcal{L}(\{\!\{A_{\mathtt{p}}\}\!\}_{\mathtt{p}\in\mathcal{P}}) \cap \Sigma^{\omega}$. ◀

▶ **Lemma 22.** *Let $w \in \Sigma^{\infty}$ be channel-compliant. Then, $w \sim w'$ iff w' is channel-compliant and $w\Downarrow_{\Sigma_{\mathtt{p}}} = w'\Downarrow_{\Sigma_{\mathtt{p}}}$ for all roles $\mathtt{p} \in \mathcal{P}$.*

Proof. We use the characterisation of $\sim$ using dependence graphs [27]. For a word w and a letter $a \in \Sigma$ that appears in w, let (a, i) denote the ith occurrence of a in w. Define the dependence graph (V, E, λ), where $V = \{(a, i) \mid a \in \Sigma, i \geq 1\}$, $E = \{((a, i), (b, j)) \mid a$ and b cannot be swapped and (a, i) occurs before (b, j) in $w\}$, and $\lambda(a, i) = a$ for all $a \in \Sigma$, $i \geq 1$. A fundamental result of trace theory states that $w \sim w'$ iff they have isomorphic dependence graphs. We observe that for two channel compliant words, the ordering of the letters on each $\Sigma_{\mathtt{p}}$ for $\mathtt{p} \in \mathcal{P}$ ensures isomorphic dependence graphs, since the ordering of receives is thus fixed. ◀

Separating Sessions Smoothly

Simon Fowler
University of Glasgow, UK

Wen Kokke
The University of Edinburgh, UK

Ornela Dardha
University of Glasgow, UK

Sam Lindley
The University of Edinburgh, UK

J. Garrett Morris
The University of Iowa, Iowa City, IA, USA

Abstract

This paper introduces Hypersequent GV (HGV), a modular and extensible core calculus for functional programming with session types that enjoys deadlock freedom, confluence, and strong normalisation. HGV exploits hyper-environments, which are collections of type environments, to ensure that structural congruence is type preserving. As a consequence we obtain a tight operational correspondence between HGV and HCP, a hypersequent-based process-calculus interpretation of classical linear logic. Our translations from HGV to HCP and vice-versa both preserve and reflect reduction. HGV scales smoothly to support Girard's Mix rule, a crucial ingredient for channel forwarding and exceptions.

2012 ACM Subject Classification Software and its engineering → Functional languages; Theory of computation → Linear logic

Keywords and phrases session types, hypersequents, linear lambda calculus

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2021.36

Related Version *Extended Version*: `https://arxiv.org/abs/2105.08996`

Funding Work supported by EPSRC grants EP/K034413/1 (ABCD), EP/T014628/1 (STARDUST), EP/L01503X/1 (CDT PPar), ERC Consolidator Grant 682315 (Skye), UKRI Future Leaders Fellowship MR/T043830/1 (EHOP), a UK Government ISCF Metrology Fellowship grant, EU HORIZON 2020 MSCA RISE project 778233 (BehAPI), and NSF grant CCF-2044815.

Acknowledgements We thank the anonymous reviewers for their helpful comments and suggestions.

1 Introduction

Session types [18, 45, 19] are types used to verify communication protocols in concurrent and distributed systems: just as data types rule out dividing an integer by a string, session types rule out sending along an input channel. Session types originated in process calculi, but there is a gap between process calculi, which model the evolving state of concurrent systems, and the descriptions of these systems in typical programming languages. This paper addresses two foundations for session types: (1) a session-typed concurrent lambda calculus called GV [30], intended to be a modular and extensible basis for functional programming languages with session types; and, (2) a session-typed process calculus called CP [51], with a propositions-as-types correspondence to classical linear logic (CLL) [17].

Processes in CP correspond exactly to proofs in CLL and deadlock freedom follows from cut-elimination for CLL. However, while CP is strongly tied to CLL, at the same time it departs from π-calculus. Independent π-calculus features can only appear in combination in

32nd International Conference on Concurrency Theory (CONCUR 2021).
Editors: Serge Haddad and Daniele Varacca; Article No. 36; pp. 36:1–36:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

CP: CP combines name restriction with parallel composition ($(\nu x)(P \parallel Q)$), corresponding to CLL's cut rule, and combines sending (of bound names only) with parallel composition ($x[y].(P \parallel Q)$), corresponding to CLL's tensor rule. This results in a proliferation of process constructors and prevents the use of standard techniques from concurrency theory, such as labelled-transition semantics and bisimulation, since the expected transitions give rise to ill-typed terms: for example, we cannot write the expected transition rule for output, $x[y].(P \parallel Q) \xrightarrow{x[y]} P \parallel Q$, since $P \parallel Q$ is not a valid CP process. A similar issue arises when attempting to design a synchronisation transition rule for bound output; see [27] for a detailed discussion. Inspired by Carbone *et al.* [10] who used hypersequents [4] to give a logical grounding to choreographic programming languages [33], Hypersequent CP (HCP) [26, 27, 34] restores the independence of these features by factoring out parallel composition into a standalone construct while retaining the close correspondence with CLL proofs. HCP typing reasons about collections of processes using collections of type environments (or *hyper-environments*).

GV extends linear λ-calculus with constants for session-typed communication. Following Gay and Vasconcelos [16], Lindley and Morris [30] describe GV's semantics by combining a reduction relation on single terms, following standard λ-calculus rules, and a reduction relation on concurrent configurations of terms, following standard π-calculus rules. They then give a semantic characterisation of deadlocked processes, an extrinsic [42] type system for configurations, and show that well-typed configurations are deadlock-free. There is, however, a large fly in this otherwise smooth ointment: process equivalence does not preserve typing. As a result, it is not enough for Lindley and Morris to show progress and preservation for well-typed configurations; instead, they must show progress and preservation for *all* configurations *equivalent to* well-typed configurations. This not only complicates the metatheory of GV, but the burden is inherited by any effort to build on GV's account of concurrency [14].

In this paper, we show that using hyper-environments in the typing of configurations enables a metatheory for GV that, compared to that of Lindley and Morris, is simpler, is more general, and as a result is easier to use and easier to extend. Hypersequent GV (HGV) repairs the treatment of process equivalence – equivalent configurations are equivalently typeable – and avoids the need for formal gimmickry connecting name restriction and parallel composition. HGV admits standard semantic techniques for concurrent programs: we use bisimulation to show that our translations both preserve *and reflect* reduction, whereas Lindley and Morris show only that their translations between GV and CP preserve reduction as well as resorting to weak explicit substitutions [28]. HGV is also more easily extensible: we outline three examples, including showing that HGV naturally extends to disconnected sets of communication processes, without any change to the proof of deadlock freedom, and that it serves as a simpler foundation for existing work on exceptions in GV [14].

Contributions. The paper contributes the following:

- Section 3 introduces Hypersequent GV (HGV), a modular and extensible core calculus for functional programming with session types which uses hyper-environments to ensure that structural congruence is type preserving.
- Section 4 shows that every well-typed GV configuration is also a well-typed HGV configuration, and every tree-structured HGV configuration is equivalent to a well-typed GV configuration.
- Section 5 gives a tight operational correspondences between HGV and HCP via translations in both directions that preserve and reflect reduction.
- Section 6 demonstrates the extensibility of HGV through: (1) unconnected processes, (2) a simplified treatment of forwarding, and (3) an improved foundation for exceptions.

Section 2 reviews GV and its metatheory, Section 7 discusses related work, and Section 8 concludes and discusses future work.

2 The Equivalence Embroglio

GV programs are deadlock free, which GV ensures by restricting process structures to trees. A *process structure* is an undirected graph where nodes represent processes and edges represent channels shared between the connected nodes. Session-typed programs with an acyclic process structure are deadlock-free by construction. We illustrate this with a session-typed vending machine example written in GV.

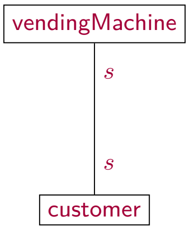
▶ **Example 2.1.** Consider the session type of a vending machine below, which sells candy bars and lollipops. If the vending machine is free, the customer can press ① to receive a candy bar or ② to receive a lollipop. If the vending machine is busy, the session ends.

$$\mathsf{VendingMachine} \triangleq \oplus\left\{\begin{array}{lll}\mathsf{Free} & : & \&\{\textcircled{1} : !\mathsf{CandyBar}.\mathbf{end}_!, \textcircled{2} : !\mathsf{Lollipop}.\mathbf{end}_!\} \\ \mathsf{Busy} & : & \mathbf{end}_!\end{array}\right\}$$

The customer's session type is *dual*: where the vending machine sends a CandyBar, the customer receives a CandyBar, and so forth. Figure 1 shows the vending machine and customer as a GV program with its process structure.

```
let vendingMachine = λs.
  let s = select Free s in
  let s = offer s { ① ↦ send candyBar
                    ② ↦ send lollipop }
  close s
in let customer = λs.
  offer s { Free ↦ let s = select ① s in
                   let (cb, s) = recv s in
                   wait s; eat cb
            Busy ↦ wait s; hungry }
in let s = fork (λs.vendingMachine s)
in customer s
```

(a) Vending machine and customer as a GV program.



(b) Process structure of Figure 1a.

Figure 1 Example program with process structure.

GV establishes the restriction to tree-structured processes by restricting the primitive for spawning processes. In GV, **fork** has type $(S \multimap \mathbf{end}_!) \multimap \overline{S}$. It takes a closure of type $S \multimap \mathbf{end}_!$ as an argument, creates a channel with endpoints of dual types S and $\overline{S}$, spawns the closure as a new process by supplying one of the endpoints as an argument, and then returns the other endpoint. In essence, **fork** is a branching operation on the process structure: it creates a new node connected to the current node by a single edge. Linearity guarantees that the tree structure is preserved, even in the presence of higher-order channels.

Lindley and Morris [30] introduce a semantics for GV, which evaluates programs embedded in process configurations, consisting of embedded programs, flagged as main (•) or child (◦) threads, ν-binders to create new channels, and parallel compositions:

$$\mathcal{C}, \mathcal{D} ::= \bullet\, M \mid \circ\, M \mid (\nu x)\mathcal{C} \mid (\mathcal{C} \parallel \mathcal{D})$$

They introduce these process configurations together with a standard structural congruence, which allows, amongst other things, the reordering of processes using commutativity ($\mathcal{C} \parallel \mathcal{C}' \equiv \mathcal{C}' \parallel \mathcal{C}$), associativity ($\mathcal{C} \parallel (\mathcal{C}' \parallel \mathcal{C}'') \equiv (\mathcal{C} \parallel \mathcal{C}') \parallel \mathcal{C}''$), and scope extrusion ($\mathcal{C} \parallel (\nu x)\mathcal{C}' \equiv (\nu x)(\mathcal{C} \parallel \mathcal{C}')$ if $x \notin \mathrm{fv}(\mathcal{C})$). They guarantee acyclicity by defining an extrinsic type system for configurations. In particular, the type system requires that in every parallel composition $\mathcal{C} \parallel \mathcal{D}$, $\mathcal{C}$ and $\mathcal{D}$ must have exactly one channel in common, and that in a name restriction $(\nu x)\mathcal{C}$, channel x cannot be used until it is shared across a parallel composition.

These restrictions are sufficient to guarantee deadlock freedom. Unfortunately, they are not preserved by process equivalence. As Lindley and Morris write, (noting that their name restrictions bind *channels* rather than endpoint pairs, and their (νxy) abbreviates $(\nu x)(\nu y)$):

> Alas, our notion of typing is not preserved by configuration equivalence. For example, assume that $\Gamma \vdash (\nu xy)(C_1 \parallel (C_2 \parallel C_3))$, where $x \in \mathrm{fv}(C_1), y \in \mathrm{fv}(C_2)$, and $x, y \in \mathrm{fv}(C_3)$. We have that $C_1 \parallel (C_2 \parallel C_3) \equiv (C_1 \parallel C_2) \parallel C_3$, but $\Gamma \nvdash (\nu xy)((C_1 \parallel C_2) \parallel C_3)$, as both x and y must be shared between the processes $C_1 \parallel C_2$ and C_3.

As a result, standard notions of progress and preservation are not enough to guarantee deadlock freedom, as reduction sequences could include equivalence steps from well-typed to non-well-typed terms! Instead, they must prove a stronger result:

▶ **Theorem 3** (Lindley and Morris [30]). *If $\Gamma \vdash \mathcal{C}$, $\mathcal{C} \equiv \mathcal{C}'$, and $\mathcal{C}' \longrightarrow \mathcal{D}'$, then there exists $\mathcal{D}$ such that $\mathcal{D} \equiv \mathcal{D}'$ and $\Gamma \vdash \mathcal{D}$.*

This is not a one-time cost: languages based on GV must either also give up on type preservation for structural congruence [14] or admit deadlocks [20, 46].

3 Hypersequent GV

We present Hypersequent GV (HGV), a linear λ-calculus extended with session types and primitives for session-typed communication. HGV shares its syntax and static typing with GV, but uses hyper-environments for runtime typing to simplify and generalise its semantics.

Types, terms, and static typing. Types (T, U) comprise a unit type ($\mathbf{1}$), an empty type ($\mathbf{0}$), product types ($T \times U$), sum types ($T + S$), linear function types ($T \multimap U$), and session types (S).

$$T, U ::= \mathbf{1} \mid \mathbf{0} \mid T \times U \mid T + U \mid T \multimap U \mid S \qquad S ::= \,!T.S \mid ?T.S \mid \mathbf{end}_! \mid \mathbf{end}_?$$

Session types (S) comprise output ($!T.S$: send a value of type T, then behave like S), input ($?T.S$: receive a value of type T, then behave like S), and dual end types ($\mathbf{end}_!$ and $\mathbf{end}_?$). The dual endpoints restrict process structure to *trees* [51]; conflating them loosens this restriction to *forests* [3]. We let Γ, Δ range over type environments.

The terms and typing rules are given in Figure 2. The linear λ-calculus rules are standard. Each communication primitive has a type schema: **link** takes a pair of compatible endpoints and forwards all messages between them; **fork** takes a function, which is passed one endpoint (of type S) of a fresh channel yielding a new child thread, and returns the other endpoint (of type $\overline{S}$); **send** takes a pair of a value and an endpoint, sends the value over the endpoint, and returns an updated endpoint; **recv** takes an endpoint, receives a value over the endpoint, and returns the pair of the received value and an updated endpoint; and **wait** synchronises on a terminated endpoint of type $\mathbf{end}_?$. Output is dual to input, and $\mathbf{end}_!$ is dual to $\mathbf{end}_?$. Duality is involutive, i. e., $\overline{\overline{S}} = S$.

Typing rules for terms $\boxed{\Gamma \vdash M : T}$

$$\textsc{TM-Var}\ \frac{}{x : T \vdash x : T} \qquad \textsc{TM-Const}\ \frac{}{\cdot \vdash K : T} \qquad \textsc{TM-Lam}\ \frac{\Gamma, x : T \vdash M : U}{\Gamma \vdash \lambda x.M : T \multimap U} \qquad \textsc{TM-App}\ \frac{\Gamma \vdash M : T \multimap U \qquad \Delta \vdash N : T}{\Gamma, \Delta \vdash M\ N : U}$$

$$\textsc{TM-Unit}\ \frac{}{\cdot \vdash () : \mathbf{1}} \qquad \textsc{TM-LetUnit}\ \frac{\Gamma \vdash M : \mathbf{1} \qquad \Delta \vdash N : T}{\Gamma, \Delta \vdash \mathbf{let}\ () = M\ \mathbf{in}\ N : T} \qquad \textsc{TM-Pair}\ \frac{\Gamma \vdash M : T \qquad \Delta \vdash N : U}{\Gamma, \Delta \vdash (M, N) : T \times U}$$

$$\textsc{TM-LetPair}\ \frac{\Gamma \vdash M : T \times T' \qquad \Delta, x : T, y : T' \vdash N : U}{\Gamma, \Delta \vdash \mathbf{let}\ (x, y) = M\ \mathbf{in}\ N : U} \qquad \textsc{TM-Absurd}\ \frac{\Gamma \vdash M : \mathbf{0}}{\Gamma \vdash \mathbf{absurd}\ M : T} \qquad \textsc{TM-Inl}\ \frac{\Gamma \vdash M : T}{\Gamma \vdash \mathbf{inl}\ M : T + U}$$

$$\textsc{TM-Inr}\ \frac{\Gamma \vdash M : U}{\Gamma \vdash \mathbf{inr}\ M : T + U} \qquad \textsc{TM-CaseSum}\ \frac{\Gamma \vdash L : T + T' \qquad \Delta, x : T \vdash M : U \qquad \Delta, y : T' \vdash N : U}{\Gamma, \Delta \vdash \mathbf{case}\ L\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} : U}$$

Type schemas for communication primitives $\boxed{K : T}$

$$\begin{array}{lll}
\mathbf{link} : S \times \overline{S} \multimap \mathbf{end}_! & \mathbf{send} : T \times {!T.S} \multimap S & \mathbf{wait} : \mathbf{end}_? \multimap \mathbf{1} \\
\mathbf{fork} : (S \multimap \mathbf{end}_!) \multimap \overline{S} & \mathbf{recv} : {?T.S} \multimap T \times S &
\end{array}$$

Duality $\boxed{\overline{S}}$

$$\overline{!T.S} = {?T.\overline{S}} \qquad \overline{?T.S} = {!T.\overline{S}} \qquad \overline{\mathbf{end}_!} = \mathbf{end}_? \qquad \overline{\mathbf{end}_?} = \mathbf{end}_!$$

Figure 2 HGV, duality and typing rules for terms.

We write $M; N$ for $\mathbf{let}\ () = M\ \mathbf{in}\ N$, $\mathbf{let}\ x = M\ \mathbf{in}\ N$ for $(\lambda x.N)\ M$, $\lambda().M$ for $\lambda z.z; M$, and $\lambda(x, y).M$ for $\lambda z.\mathbf{let}\ (x, y) = z\ \mathbf{in}\ M$. We write $K : T$ for $\cdot \vdash K : T$ in typing derivations.

▶ Remark 3.1. We include **link** because it is convenient for the correspondence with CP, which interprets CLL's axiom as forwarding. We *can* encode **link** in GV via a type directed translation akin to CLL's *identity expansion*.

Configurations and runtime typing. Process configurations $(\mathcal{C}, \mathcal{D}, \mathcal{E})$ comprise child threads $(\circ\ M)$, the main thread $(\bullet\ M)$, link threads $(x \overset{z}{\leftrightarrow} y)$, name restrictions $((\nu xy)\mathcal{C})$, and parallel compositions $(\mathcal{C} \parallel \mathcal{D})$. We refer to a configuration of the form $\circ M$ or $x \overset{z}{\leftrightarrow} y$ as an *auxiliary thread*, and a configuration of the form $\bullet M$ as a *main thread*. We let $\mathcal{A}$ range over auxiliary threads and $\mathcal{T}$ range over all threads (auxiliary or main).

$$\phi ::= \bullet \mid \circ \qquad\qquad \mathcal{C}, \mathcal{D}, \mathcal{E} ::= \phi\ M \mid x \overset{z}{\leftrightarrow} y \mid \mathcal{C} \parallel \mathcal{D} \mid (\nu xy)\mathcal{C}$$

The configuration language is reminiscent of π-calculus processes, but has some non-standard features. Name restriction uses double binders [49] in which one name is bound to each endpoint of the channel. Link threads [31] handle forwarding. A link thread $x \overset{z}{\leftrightarrow} y$ waits for the thread connected to z to terminate before forwarding all messages between x and y.

Configuration typing departs from GV [30], exploiting *hypersequents* [4] to recover modularity and extensibility. Inspired by HCP [34, 27, 26], configurations are typed under a *hyper-environment*, a collection of disjoint type environments. We let $\mathcal{G}, \mathcal{H}$ range over hyper-environments, writing $\varnothing$ for the empty hyper-environment, $\mathcal{G} \parallel \Gamma$ for disjoint extension of $\mathcal{G}$ with type environment Γ, and $\mathcal{G} \parallel \mathcal{H}$ for disjoint concatenation of $\mathcal{G}$ and $\mathcal{H}$.

Typing rules for configurations $\boxed{\mathcal{G} \vdash \mathcal{C} : R}$

$$\textsc{TC-New}\ \dfrac{\mathcal{G} \parallel \Gamma, x : S \parallel \Delta, y : \overline{S} \vdash \mathcal{C} : R}{\mathcal{G} \parallel \Gamma, \Delta \vdash (\nu xy)\mathcal{C} : R} \qquad \textsc{TC-Par}\ \dfrac{\mathcal{G} \vdash \mathcal{C} : R \qquad \mathcal{H} \vdash \mathcal{D} : R'}{\mathcal{G} \parallel \mathcal{H} \vdash \mathcal{C} \parallel \mathcal{D} : R \sqcap R'}$$

$$\textsc{TC-Main}\ \dfrac{\Gamma \vdash M : T}{\Gamma \vdash \bullet\, M : \bullet\, T} \qquad \textsc{TC-Child}\ \dfrac{\Gamma \vdash M : \mathbf{end}_!}{\Gamma \vdash \circ\, M : \circ} \qquad \textsc{TC-Link}\ \dfrac{}{x : S, y : \overline{S}, z : \mathbf{end}_? \vdash x \overset{z}{\leftrightarrow} y : \circ}$$

Configuration types **Configuration type combination** $\boxed{R \sqcap R'}$

$$R ::= \circ \mid \bullet\, T \qquad\qquad \bullet\, T \sqcap \circ = \bullet\, T \qquad \circ \sqcap \bullet\, T = \bullet\, T \qquad \circ \sqcap \circ = \circ$$

Figure 3 HGV, typing rules for configurations.

Structural congruence $\boxed{\mathcal{C} \equiv \mathcal{D}}$

SC-ParAssoc	$\mathcal{C} \parallel (\mathcal{D} \parallel \mathcal{E}) \equiv (\mathcal{C} \parallel \mathcal{D}) \parallel \mathcal{E}$	SC-ParComm	$\mathcal{C} \parallel \mathcal{D} \equiv \mathcal{D} \parallel \mathcal{C}$
SC-NewComm	$(\nu xy)(\nu zw)\mathcal{C} \equiv (\nu zw)(\nu xy)\mathcal{C}$	SC-NewSwap	$(\nu xy)\mathcal{C} \equiv (\nu yx)\mathcal{C}$
SC-ScopeExt	$(\nu xy)(\mathcal{C} \parallel \mathcal{D}) \equiv \mathcal{C} \parallel (\nu xy)\mathcal{D}$, if $x, y \notin \mathrm{fv}(\mathcal{C})$	SC-LinkComm	$x \overset{z}{\leftrightarrow} y \equiv y \overset{z}{\leftrightarrow} x$

Configuration reduction $\boxed{\mathcal{C} \longrightarrow \mathcal{D}}$

E-Reify-Fork	$F[\mathbf{fork}\ V]$	$\longrightarrow$	$(\nu xx')(F[x] \parallel \circ\,(V\ x'))$, where x, x' fresh
E-Reify-Link	$F[\mathbf{link}\ (x, y)]$	$\longrightarrow$	$(\nu zz')(x \overset{z}{\leftrightarrow} y \parallel F[z'])$, where z, z' fresh
E-Comm-Link	$(\nu zz')(\nu xx')(x \overset{z}{\leftrightarrow} y \parallel \circ\, z' \parallel \phi\ M)$	$\longrightarrow$	$\phi\ (M\{y/x'\})$
E-Comm-Send	$(\nu xy)(F[\mathbf{send}\ (V, x)] \parallel F'[\mathbf{recv}\ y])$	$\longrightarrow$	$(\nu xy)(F[x] \parallel F'[(V, y)])$
E-Comm-Close	$(\nu xy)(\circ\, y \parallel F[\mathbf{wait}\ x])$	$\longrightarrow$	$F[()]$

$$\textsc{E-Res}\ \dfrac{\mathcal{C} \longrightarrow \mathcal{C}'}{(\nu xy)\mathcal{C} \longrightarrow (\nu xy)\mathcal{C}'} \qquad \textsc{E-Par}\ \dfrac{\mathcal{C} \longrightarrow \mathcal{C}'}{\mathcal{C} \parallel \mathcal{D} \longrightarrow \mathcal{C}' \parallel \mathcal{D}} \qquad \textsc{E-Equiv}\ \dfrac{\mathcal{C} \equiv \mathcal{C}' \quad \mathcal{C}' \longrightarrow \mathcal{D}' \quad \mathcal{D}' \equiv \mathcal{D}}{\mathcal{C} \longrightarrow \mathcal{D}} \qquad \textsc{E-Lift-M}\ \dfrac{M \longrightarrow_{\mathsf{M}} M'}{F[M] \longrightarrow F[M']}$$

Figure 4 HGV, configuration reduction.

The typing rules for configurations are given in Figure 3. Rules TC-New and TC-Par are key to deadlock freedom: TC-New joins two disjoint configurations with a new channel, and merges their type environments; TC-Par combines two disjoint configurations, and registers their disjointness by separating their type environments in the hyper-environment. Rules TC-Main, TC-Child, and TC-Link type main, child, and link threads, respectively; all three require a singleton hyper-environment. A configuration has type $\circ$ if it has no main thread, and $\bullet\ T$ if it has a main thread of type T. The configuration type combination operator ensures that a well-typed configuration has at most one main thread.

Operational semantics. HGV values (U, V, W), evaluation contexts (E), and term reduction rules ($\longrightarrow_{\mathsf{M}}$) define a standard call-by-value, left-to-right evaluation strategy. A closed term either reduces to a value or is blocked on a communication action.

Figure 4 gives the configuration reduction rules. Thread contexts (F) extend evaluation contexts to threads, i. e., $F ::= \phi\ E$. The structural congruence rules are standard apart from SC-LinkComm, which ensures links are undirected, and SC-NewSwap, which swaps names in double binders. The concurrent behaviour of HGV is given by a nondeterministic reduction relation ($\longrightarrow$) on configurations. The first two rules, E-Reify-Fork and E-Reify-Link, create child and link threads, respectively. The next three rules, E-Comm-Link, E-Comm-Send, and E-Comm-Close perform communication actions. The final four rules enable reduction under name restriction and parallel composition, rewriting by structural congruence, and term reduction in threads. Two rules handle links: E-Reify-Link creates a new *link thread* $x \overset{z}{\leftrightarrow} y$ which blocks on z of type $\mathbf{end}_?$, one endpoint of a fresh channel. The other endpoint, z' of type $\mathbf{end}_!$, is placed in the evaluation context of the parent thread. When z' terminates a child thread, E-Comm-Link performs forwarding by substitution.

Choice. Internal and external choice are encoded with sum types and session delegation [22, 13]. Prior encodings of choice in GV [30] are asynchronous. To encode synchronous choice we add a dummy synchronisation before exchanging the value of sum type, as follows:

$$\begin{aligned}
S \oplus S' &\triangleq\ !\mathbf{1}.!(\overline{S_1} + \overline{S_2}).\mathbf{end}_! \\
S \,\&\, S' &\triangleq\ ?\mathbf{1}.?(S_1 + S_2).\mathbf{end}_? \\
\oplus\{\} &\triangleq\ !\mathbf{1}.!\mathbf{0}.\mathbf{end}_! \\
\&\{\} &\triangleq\ ?\mathbf{1}.?\mathbf{0}.\mathbf{end}_?
\end{aligned}$$

$$\begin{aligned}
\mathbf{select}\ \ell &\triangleq \lambda x. \begin{pmatrix} \mathbf{let}\ x = \mathbf{send}\ ((), x)\ \mathbf{in} \\ \mathbf{fork}\ (\lambda y.\mathbf{send}\ (\ell\ y, x)) \end{pmatrix} \\
\mathbf{offer}\ L\ \{\mathbf{inl}\ x \mapsto M; \mathbf{inr}\ y \mapsto N\} & \\
&\triangleq \mathbf{let}\ ((), z) = \mathbf{recv}\ L\ \mathbf{in}\ \mathbf{let}\ (w, z) = \mathbf{recv}\ z \\
&\ \mathbf{in}\ \mathbf{wait}\ z; \mathbf{case}\ w\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\} \\
\mathbf{offer}\ L\ \{\} &\triangleq \mathbf{let}\ ((), c) = \mathbf{recv}\ L\ \mathbf{in}\ \mathbf{let}\ (z, c) = \mathbf{recv}\ c \\
&\ \mathbf{in}\ \mathbf{wait}\ c; \mathbf{absurd}\ \ z
\end{aligned}$$

HGV enjoys type preservation, deadlock freedom, confluence, and strong normalisation (details in the extended version). Here we outline where the metatheory diverges from GV.

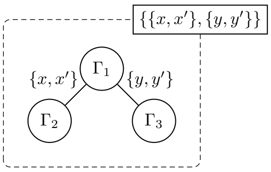
Preservation. Hyper-environments enable type preservation under structural congruence, which significantly simplifies the metatheory compared to GV.

▶ **Theorem 3.2** (Preservation).

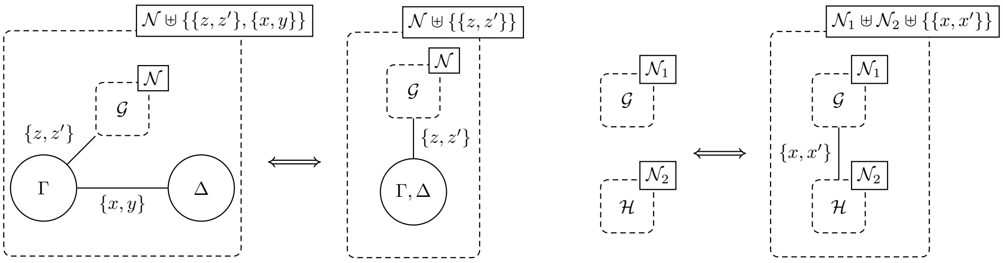
1. *If* $\mathcal{G} \vdash \mathcal{C} : R$ *and* $\mathcal{C} \equiv \mathcal{D}$*, then* $\mathcal{G} \vdash \mathcal{D} : R$*.*
2. *If* $\mathcal{G} \vdash \mathcal{C} : R$ *and* $\mathcal{C} \longrightarrow \mathcal{D}$*, then* $\mathcal{G} \vdash \mathcal{D} : R$*.*

Abstract process structures. Unlike in GV, in HGV we cannot rely on the fact that exactly one channel is split over each parallel composition. Instead, we introduce the notion of an *abstract process structure* (APS). An APS is a graph defined over a hyper-environment $\mathcal{G}$ and a set of undirected pairs of co-names (a *co-name set*) $\mathcal{N}$ drawn from the names in $\mathcal{G}$. The nodes of an APS are the type environments in $\mathcal{G}$. Each edge is labelled by a distinct co-name pair $\{x_1, x_2\} \in \mathcal{N}$, such that $x_1 : S \in \Gamma_1$ and $x_2 : \overline{S} \in \Gamma_2$.

▶ **Example 3.3.** Let $\mathcal{G} = \Gamma_1 \parallel \Gamma_2 \parallel \Gamma_3$, where $\Gamma_1 = x : S_1, y : S_2$, $\Gamma_2 = x' : \overline{S_1}, z : T$, and $\Gamma_3 = y' : \overline{S_2}$, and suppose $\mathcal{N} = \{\{x, x'\}, \{y, y'\}\}$. The APS for $\mathcal{G}$ and $\mathcal{N}$ is illustrated below.



A key feature of HGV is a subformula principle, which states that all hyper-environments arising in the derivation of an HGV program are tree-structured. We write $\mathsf{Tree}(\mathcal{G}, \mathcal{N})$ to denote that the APS for $\mathcal{G}$ with respect to $\mathcal{N}$ is tree-structured. An HGV program $\bullet\, M$ has a single type environment, so is tree-structured; the same goes for child and link threads. Read bottom-up TC-New and TC-Par preserve tree structure (see the extended version for formal statements), which is illustrated by the following two pictures.



Tree canonical form. We now define a canonical form for configurations that captures the tree structure of an APS. Tree canonical form enables a succinct statement of *open progress* (Lemma 3.8) and a means for embedding HGV in GV (Lemma 4.5).

▶ **Definition 3.4** (Tree canonical form). *A configuration $\mathcal{C}$ is in* tree canonical form *if it can be written: $(\nu x_1 y_1)(\mathcal{A}_1 \parallel \cdots \parallel (\nu x_n y_n)(\mathcal{A}_n \parallel \phi N) \cdots)$ where $x_i \in \mathrm{fv}(\mathcal{A}_i)$ for $1 \le i \le n$.*

▶ **Theorem 3.5** (Tree canonical form). *If $\Gamma \vdash \mathcal{C} : R$, then there exists some $\mathcal{D}$ such that $\mathcal{C} \equiv \mathcal{D}$ and $\mathcal{D}$ is in tree canonical form.*

▶ **Lemma 3.6.** *If $\Gamma_1 \parallel \cdots \parallel \Gamma_n \vdash \mathcal{C} : R$, then there exist $R_1, \ldots, R_n$ and $\mathcal{D}_1, \ldots, \mathcal{D}_n$ such that $R = R_1 \sqcap \cdots \sqcap R_n$ and $\mathcal{C} \equiv \mathcal{D}_1 \parallel \cdots \parallel \mathcal{D}_n$ and $\Gamma_i \vdash \mathcal{D}_i : R_i$ for each i.*

It follows from Theorem 3.5 and Lemma 3.6 that any well-typed HGV configuration can be written as a forest of independent configurations in tree canonical form.

Progress and Deadlock Freedom.

▶ **Definition 3.7** (Blocked thread). *We say that thread $\mathcal{T}$ is* blocked on variable *z, written* blocked$(\mathcal{T}, z)$*, if either: $\mathcal{T} = \circ\, z$; $\mathcal{T} = x \overset{z}{\leftrightsquigarrow} y$, for some x, y; or $\mathcal{T} = F[N]$ for some F, where N is **send** (V, z), **recv** z, or **wait** z.*

We let Ψ range over type environments containing only session-typed variables, i. e., $\Psi ::= \cdot \mid \Psi, x : S$, which lets us reason about configurations that are closed except for runtime names. Using Lemma 3.6 we obtain *open progress* for configurations with free runtime names.

▶ **Lemma 3.8** (Open Progress). *Suppose $\Psi \vdash \mathcal{C} : T$ where $\mathcal{C} = (\nu x_1 y_1)(\mathcal{A}_1 \parallel \cdots \parallel (\nu x_n y_n)(\mathcal{A}_n \parallel \phi N) \cdots)$ is in tree canonical form. Either $\mathcal{C} \longrightarrow \mathcal{D}$ for some $\mathcal{D}$, or:*

1. *For each $\mathcal{A}_i$ $(1 \le i \le n)$,* blocked$(\mathcal{A}_i, z)$ *for some $z \in \{x_i\} \cup \{y_j \mid 1 \le j < i\} \cup \mathrm{fv}(\Psi)$*
2. *Either N is a value or* blocked$(\phi N, z)$ *for some $z \in \{y_i \mid 1 \le i \le n\} \cup \mathrm{fv}(\Psi)$*

For closed configurations, we obtain a tighter result. If a closed configuration cannot reduce, then each auxiliary thread must either be a value, or be blocked on its neighbouring endpoint.

Finally, for *ground configurations*, where the main thread does not return a runtime name or capture a runtime name in a closure, we obtain a yet tighter result, *global progress*, which implies deadlock freedom [8].

▶ **Definition 3.9** (Ground configuration). *A configuration $\mathcal{C}$ is a* ground configuration *if $\cdot \vdash \mathcal{C} : T$, $\mathcal{C}$ is in canonical form, and T does not contain session types or function types.*

▶ **Theorem 3.10** (Global progress). *Suppose $\mathcal{C}$ is a ground configuration. Either there exists some $\mathcal{D}$ such that $\mathcal{C} \longrightarrow \mathcal{D}$, or $\mathcal{C} = \bullet V$ for some value V.*

Typing rules for configurations $\boxed{\Gamma \vdash_{\mathsf{GV}} \mathcal{C} : T}$

$$\textsc{TG-New}\;\frac{\Gamma, \langle x, y\rangle : S^{\sharp} \vdash_{\mathsf{GV}} \mathcal{C} : R}{\Gamma \vdash_{\mathsf{GV}} (\nu xy)\mathcal{C} : R} \qquad \textsc{TG-Connect}_1\;\frac{\Gamma_1, x : S \vdash_{\mathsf{GV}} \mathcal{C} : R \quad \Gamma_2, y : \overline{S} \vdash_{\mathsf{GV}} \mathcal{D} : R'}{\Gamma_1, \Gamma_2, \langle x, y\rangle : S^{\sharp} \vdash_{\mathsf{GV}} \mathcal{C} \parallel \mathcal{D} : R \sqcap R'}$$

$$\textsc{TG-Connect}_2\;\frac{\Gamma_1, y : \overline{S} \vdash_{\mathsf{GV}} \mathcal{C} : R \quad \Gamma_2, x : S \vdash_{\mathsf{GV}} \mathcal{D} : R'}{\Gamma_1, \Gamma_2, \langle x, y\rangle : S^{\sharp} \vdash_{\mathsf{GV}} \mathcal{C} \parallel \mathcal{D} : R \sqcap R'}$$

$$\textsc{TG-Child}\;\frac{\Gamma \vdash_{\mathsf{GV}} M : \mathbf{end}_!}{\Gamma \vdash_{\mathsf{GV}} \circ M : \circ} \qquad \textsc{TG-Main}\;\frac{\Gamma \vdash_{\mathsf{GV}} M : T}{\Gamma \vdash_{\mathsf{GV}} \bullet M : \bullet\, T} \qquad \textsc{TG-Link}\;\frac{}{x : S, y : \overline{S}, z : \mathbf{end}_? \vdash_{\mathsf{GV}} x \overset{z}{\leftrightarrow} y : \circ}$$

Figure 5 GV, typing rules for configurations.

4 Relation between HGV and GV

In this section, we show that well-typed GV configurations are well-typed HGV configurations, and well-typed HGV configurations with tree structure are well-typed GV configuration.

GV. HGV and GV share a common term language and reduction semantics, so only differ in their runtime typing rules. Figure 5 gives the runtime typing rules for GV. We adapt the rules to use a double-binder formulation to concentrate on the essence of the relationship with HGV, but it is trivial to translate GV with single binders into GV with double binders.

We require a pseudo-type $S^{\sharp}$, which types un-split channels. Un-split channels cannot appear in terms. Rule TG-New types a name restriction $(\nu xy)\mathcal{C}$, adding $\langle x, y\rangle : S^{\sharp}$ to the type environment, which along with TG-Connect$_1$ and TG-Connect$_2$ ensures that a session channel of type S will be split into endpoints x and y over a parallel composition, in turn enforcing a tree process structure. The remaining typing rules are as in HGV.

Embedding GV into HGV. Every well-typed open GV configuration is also a well-typed HGV configuration.

▶ **Definition 4.1** (Flattening). *Flattening, written $\downarrow$, converts GV type environments and HGV hyper-environments into HGV environments.*

$$\begin{array}{lcl@{\qquad\qquad}lcl}
\downarrow \cdot & = & \cdot & \downarrow \varnothing & = & \varnothing \\
\downarrow(\Gamma, \langle x, x'\rangle : S^{\sharp}) & = & \downarrow\Gamma, x : S, x' : \overline{S} & \downarrow(\mathcal{G} \parallel \Gamma) & = & \downarrow\mathcal{G}, \Gamma \\
\downarrow(\Gamma, x : T) & = & \downarrow\Gamma, x : T & & &
\end{array}$$

▶ **Definition 4.2** (Splitting). *Splitting converts GV typing environments into hyper-environments. Given channels $\{\langle x_i, x'_i\rangle : S^{\sharp}_i\}_{i \in 1..n}$ in Γ, a hyper-environment $\mathcal{G}$ is a* splitting *of Γ if $\downarrow\mathcal{G} = \downarrow\Gamma$ and $\exists \Gamma_1, \ldots, \Gamma_{n+1}$ such that $\mathcal{G} = \Gamma_1 \parallel \cdots \parallel \Gamma_{n+1}$, and* $\mathsf{Tree}(\mathcal{G}, \{\{x_1, x'_1\}, \ldots, \{x_n, x'_n\}\})$.

A well-typed GV configuration is typeable in HGV under a splitting of its type environment.

▶ **Theorem 4.3** (Typeability of GV configurations in HGV). *If $\Gamma \vdash_{\mathsf{GV}} \mathcal{C} : R$, then there exists some $\mathcal{G}$ such that $\mathcal{G}$ is a splitting of Γ and $\mathcal{G} \vdash \mathcal{C} : R$.*

▶ **Example 4.4.** Consider a configuration where a child thread pings the main thread:

$$(\nu xy)(\circ\,(\mathbf{send}\;(ping, x)) \parallel \bullet\,(\mathbf{let}\;((), y) = \mathbf{recv}\;y\;\mathbf{in}\;\mathbf{wait}\;y))$$

We can write a GV typing derivation as follows:

$$\dfrac{\dfrac{x : !\mathbf{1}.\mathbf{end}_{!}, \mathit{ping} : \mathbf{1} \vdash_{\mathsf{GV}} \circ\,(\mathbf{send}\ (\mathit{ping}, x)) : \circ \qquad y : ?\mathbf{1}.\mathbf{end}_{?} \vdash_{\mathsf{GV}} \bullet\,(\mathbf{let}\ ((), y) = \mathbf{recv}\ y\ \mathbf{in}\ \mathbf{wait}\ y) : \bullet\,\mathbf{1}}{\langle x, y\rangle : !\mathbf{1}.\mathbf{end}_{!}{}^{\sharp}, \mathit{ping} : \mathbf{1} \vdash_{\mathsf{GV}} (\nu xy)(\circ(\mathbf{send}\ (\mathit{ping}, x)) \parallel \bullet(\mathbf{let}\ ((), y) = \mathbf{recv}\ y\ \mathbf{in}\ \mathbf{wait}\ y)) : \mathbf{1}}}{\mathit{ping} : \mathbf{1} \vdash_{\mathsf{GV}} (\nu xy)(\circ(\mathbf{send}\ (\mathit{ping}, x)) \parallel \bullet(\mathbf{let}\ ((), y) = \mathbf{recv}\ y\ \mathbf{in}\ \mathbf{wait}\ y)) : \mathbf{1}}$$

The corresponding HGV derivation is:

$$\dfrac{\dfrac{x : !\mathbf{1}.\mathbf{end}_{!}, \mathit{ping} : \mathbf{1} \vdash \circ\,(\mathbf{send}\ (\mathit{ping}, x)) : \circ \qquad y : ?\mathbf{1}.\mathbf{end}_{?} \vdash \bullet\,(\mathbf{let}\ ((), y) = \mathbf{recv}\ y\ \mathbf{in}\ \mathbf{wait}\ y) : \bullet\,\mathbf{1}}{x : !\mathbf{1}.\mathbf{end}_{!}, \mathit{ping} : \mathbf{1} \parallel y : ?\mathbf{1}.\mathbf{end}_{?} \vdash (\nu xy)(\circ(\mathbf{send}\ (\mathit{ping}, x)) \parallel \bullet(\mathbf{let}\ ((), y) = \mathbf{recv}\ y\ \mathbf{in}\ \mathbf{wait}\ y)) : \bullet\,\mathbf{1}}}{\mathit{ping} : \mathbf{1} \vdash (\nu xy)(\circ(\mathbf{send}\ (\mathit{ping}, x)) \parallel \bullet(\mathbf{let}\ ((), y) = \mathbf{recv}\ y\ \mathbf{in}\ \mathbf{wait}\ y)) : \bullet\,\mathbf{1}}$$

Note that $x : !\mathbf{1}.\mathbf{end}_{!}, \mathit{ping} : \mathbf{1} \parallel y : ?\mathbf{1}.\mathbf{end}_{?}$ is a splitting of $\langle x, y\rangle : (!\mathbf{1}.\mathbf{end}_{!})^{\sharp}, \mathit{ping} : \mathbf{1}$.

Translating HGV to GV. As we saw in §2, unlike in HGV, equivalence in GV is not type-preserving. It follows that HGV types strictly more processes than GV. Let us revisit Lindley and Morris' example from §1 (adapted to use double-binders), where $\Gamma_1, \Gamma_2, \Gamma_3 \vdash_{\mathsf{GV}} (\nu x x')(\nu y y')(\mathcal{C} \parallel (\mathcal{D} \parallel \mathcal{E})) : R_1 \sqcap R_2 \sqcap R_3$ with $\Gamma_1, x : S \vdash_{\mathsf{GV}} \mathcal{C} : R_1$, $\Gamma_2, y : S' \vdash_{\mathsf{GV}} \mathcal{D} : R_2$, and $\Gamma_3, x' : \overline{S}, y' : \overline{S'} \vdash_{\mathsf{GV}} \mathcal{E} : R_3$.

The structurally-equivalent term $(\nu x x')(\nu y y')((\mathcal{C} \parallel \mathcal{D}) \parallel \mathcal{E})$ is not typeable in GV, since we cannot split both channels over a single parallel composition:

$$\dfrac{\dfrac{\dfrac{\Gamma_1, \Gamma_2, x : S \not\vdash_{\mathsf{GV}} \mathcal{C} \parallel \mathcal{D} : R_1 \sqcap R_2 \qquad \Gamma_3, x' : \overline{S}, \langle y, y'\rangle : S'^{\sharp} \not\vdash_{\mathsf{GV}} \mathcal{E} : R_3}{\Gamma_1, \Gamma_2, \Gamma_3, \langle x, x'\rangle : S^{\sharp}, \langle y, y'\rangle : S'^{\sharp} \not\vdash_{\mathsf{GV}} (\mathcal{C} \parallel \mathcal{D}) \parallel \mathcal{E} : R_1 \sqcap R_2 \sqcap R_3}}{\Gamma_1, \Gamma_2, \Gamma_3, \langle x, x'\rangle : S^{\sharp} \not\vdash_{\mathsf{GV}} (\nu y y')((\mathcal{C} \parallel \mathcal{D}) \parallel \mathcal{E}) : R_1 \sqcap R_2 \sqcap R_3}}{\Gamma_1, \Gamma_2, \Gamma_3 \not\vdash_{\mathsf{GV}} (\nu x x')(\nu y y')((\mathcal{C} \parallel \mathcal{D}) \parallel \mathcal{E}) : R_1 \sqcap R_2 \sqcap R_3}$$

However, we *can* type this process in HGV:

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma_1, x : S \vdash \mathcal{C} : R_1 \qquad \Gamma_2, y : S' \vdash \mathcal{D} : R_2}{\Gamma_1, x : S \parallel \Gamma_2, y : S' \vdash \mathcal{C} \parallel \mathcal{D} : R_1 \sqcap R_2} \qquad \Gamma_3, x' : \overline{S}, y' : \overline{S'} \vdash \mathcal{E} : R_3}{\Gamma_1, x : S \parallel \Gamma_2, y : S' \parallel \Gamma_3, x' : \overline{S}, y' : \overline{S'} \vdash (\mathcal{C} \parallel \mathcal{D}) \parallel \mathcal{E} : R_1 \sqcap R_2 \sqcap R_3}}{\Gamma_1, x : S \parallel \Gamma_2, \Gamma_3, x' : \overline{S} \vdash (\nu y y')((\mathcal{C} \parallel \mathcal{D}) \parallel \mathcal{E}) : R_1 \sqcap R_2 \sqcap R_3}}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash (\nu x x')(\nu y y')((\mathcal{C} \parallel \mathcal{D}) \parallel \mathcal{E}) : R_1 \sqcap R_2 \sqcap R_3}$$

Although HGV types more processes, every well-typed HGV configuration typeable under a singleton hyper-environment Γ is *equivalent* to a well-typed GV configuration, which we show using tree canonical forms.

▶ **Lemma 4.5.** *Suppose $\Gamma \vdash \mathcal{C} : R$ where $\mathcal{C}$ is in tree canonical form. Then, $\Gamma \vdash_{\mathsf{GV}} \mathcal{C} : R$.*

▶ Remark 4.6. It is not the case that every HGV configuration typeable under an *arbitrary* hyper-environment $\mathcal{H}$ is equivalent to a well-typed GV configuration. This is because open HGV configurations can form *forest* process structures, whereas (even open) GV configurations must form a *tree* process structure.

Since we can write all well-typed HGV configurations in canonical form, and HGV tree canonical forms are typeable in GV, it follows that every well-typed HGV configuration typeable under a single type environment is equivalent to a well-typed GV configuration.

▶ **Corollary 4.7.** *If $\Gamma \vdash \mathcal{C} : R$, then there exists some $\mathcal{D}$ such that $\mathcal{C} \equiv \mathcal{D}$ and $\Gamma \vdash_{\mathsf{GV}} \mathcal{D} : R$.*

5 Relation between HGV and HCP

In this section, we explore two translations, from HGV to HCP and from HCP to HGV, together with their operational correspondences.

Typing rules for processes $\boxed{P \vdash \mathcal{G}}$

$$\frac{}{x \leftrightarrow^{A} y \vdash x : A, y : A^{\perp}}\ \textsc{TP-Link} \qquad \frac{P \vdash \mathcal{G} \parallel \Gamma, x : A \parallel \Delta, y : A^{\perp}}{(\nu x y)P \vdash \mathcal{G} \parallel \Gamma, \Delta}\ \textsc{TP-New} \qquad \frac{P \vdash \mathcal{G} \qquad Q \vdash \mathcal{H}}{P \parallel Q \vdash \mathcal{G} \parallel \mathcal{H}}\ \textsc{TP-Par} \qquad \frac{}{\mathbf{0} \vdash \varnothing}\ \textsc{TP-Halt} \qquad \frac{P \vdash \varnothing}{x[].P \vdash x : \mathbf{1}}\ \textsc{TP-Close}$$

$$\frac{P \vdash \Gamma}{x().P \vdash \Gamma, x : \bot}\ \textsc{TP-Wait} \qquad \frac{P \vdash \Gamma, y : A \parallel \Delta, x : B}{x[y].P \vdash \Gamma, \Delta, x : A \otimes B}\ \textsc{TP-Send} \qquad \frac{P \vdash \Gamma, y : A, x : B}{x(y).P \vdash \Gamma, x : A \parr B}\ \textsc{TP-Recv} \qquad \frac{}{x \triangleright \{\} \vdash \Gamma, x : \top}\ \textsc{TP-Offer-Absurd}$$

$$\frac{P \vdash \Gamma, x : A}{x \triangleleft \mathrm{inl}.P \vdash \Gamma, x : A \oplus B}\ \textsc{TP-Select-Inl} \qquad \frac{P \vdash \Gamma, x : B}{x \triangleleft \mathrm{inr}.P \vdash \Gamma, x : A \oplus B}\ \textsc{TP-Select-Inr} \qquad \frac{P \vdash \Gamma, x : A \qquad Q \vdash \Gamma, x : B}{x \triangleright \{\mathrm{inl} : P; \mathrm{inr} : Q\} \vdash \Gamma, x : A \mathbin{\&} B}\ \textsc{TP-Offer}$$

Duality $\boxed{A^{\perp}}$

$$\begin{array}{lclclclclclcl}
(A \otimes B)^{\perp} & = & A^{\perp} \parr B^{\perp} & (\mathbf{1})^{\perp} & = & \bot & (A \oplus B)^{\perp} & = & A^{\perp} \mathbin{\&} B^{\perp} & (\mathbf{0})^{\perp} & = & \top \\
(A \parr B)^{\perp} & = & A^{\perp} \otimes B^{\perp} & (\bot)^{\perp} & = & \mathbf{1} & (A \mathbin{\&} B)^{\perp} & = & A^{\perp} \oplus B^{\perp} & (\top)^{\perp} & = & \mathbf{0}
\end{array}$$

Figure 6 HCP, duality and typing rules for processes.

Hypersequent CP. HCP [34, 27] is a session-typed process calculus with a correspondence to CLL, which exploits hypersequents to fix extensibility and modularity issues with CP.

Types (A, B) consist of the connectives of linear logic: the multiplicative operators ($\otimes$, $\parr$) and units ($\mathbf{1}$, $\bot$) and the additive operators ($\oplus$, $\&$) and units ($\mathbf{0}$, $\top$).

$$A, B ::= \mathbf{1} \mid \bot \mid \mathbf{0} \mid \top \mid A \otimes B \mid A \parr B \mid A \oplus B \mid A \mathbin{\&} B$$

Type environments (Γ, Δ) associate names with types. Hyper-environments ($\mathcal{G}$, $\mathcal{H}$) are collections of type environments. The empty type environment and hyper-environment are written $\cdot$ and $\varnothing$, respectively. Names in type and hyper-environments must be unique and environments may be combined, written Γ, Δ and $\mathcal{G} \parallel \mathcal{H}$, only if they are disjoint.

Processes (P, Q) are a variant of the π-calculus with forwarding [44, 6], bound output [44], and double binders [49]. The syntax of processes is given by the typing rules (Figure 6), which are standard for HCP [34, 27]: $x \leftrightarrow y$ forwards messages between x and y; $(\nu x y)P$ creates a channel with endpoints x and y, and continues as P; $P \parallel Q$ composes P and Q in parallel; $\mathbf{0}$ is the terminated process; $x[y].P$ creates a new channel, outputs one endpoint over x, binds the other to y, and continues as P; $x(y).P$ receives a channel endpoint, binds it to y, and continues as P; $x[].P$ and $x().P$ close x and continue as P; $x \triangleleft \mathrm{inl}.P$ and $x \triangleleft \mathrm{inr}.P$ make a binary choice; $x \triangleright \{\mathrm{inl} : P; \mathrm{inr} : Q\}$ offers a binary choice; and $x \triangleright \{\}$ offers a nullary choice. As HCP is synchronous, the only difference between $x[y].P$ and $x(y).P$ is their typing (and similarly for $x[].P$ and $x().P$). We write *unbound* send as $x\langle y\rangle.P$ (short for $x[z].(y \leftrightarrow z \parallel P)$), and synchronisation as $\bar{x}.P$ (short for $x[z].(z[].\mathbf{0} \parallel P)$) and $x.P$ (short for $x(z).z().P$). Duality is standard and is involutive, i. e., $(A^{\perp})^{\perp} = A$.

We define a standard structural congruence ($\equiv$) similar to that of HGV, i. e., parallel composition is commutative and associative, we can commute name restrictions, swap the order of endpoints, swap links, and have scope extrusion (similar to Figure 4).

Action rules

$$\begin{array}{lllll}
\textsc{Act-Pref} & \textsc{Act-Link}_1 & \textsc{Act-Link}_2 & \textsc{Act-Off-Inl} & \textsc{Act-Off-Inr} \\
\pi.P \xrightarrow{\pi} P & x{\leftrightarrow}y \xrightarrow{x\leftrightarrow y} \mathbf{0} & x{\leftrightarrow}y \xrightarrow{y\leftrightarrow x} \mathbf{0} & x \triangleright \{\mathrm{inl} : P; \mathrm{inr} : Q\} \xrightarrow{x\triangleright\mathrm{inl}} P & x \triangleright \{\mathrm{inl} : P; \mathrm{inr} : Q\} \xrightarrow{x\triangleright\mathrm{inr}} Q
\end{array}$$

Communication Rules

$$\textsc{Tau-Alp}\ \frac{P \xrightarrow{\alpha} P'}{P \xrightarrow{\tau} P'} \qquad \textsc{Tau-Bet}\ \frac{P \xrightarrow{\beta} P'}{P \xrightarrow{\tau} P'} \qquad \textsc{Alp-Link}\ \frac{P \xrightarrow{x\leftrightarrow z} P'}{(\nu xy)P \xrightarrow{\alpha} P'\{z/y\}} \qquad \textsc{Bet-Send}\ \frac{P \xrightarrow{x[x'] \parallel y(y')} P'}{(\nu xy)P \xrightarrow{\beta} (\nu xy)(\nu x'y')P'}$$

$$\textsc{Bet-Close}\ \frac{P \xrightarrow{x[] \parallel y()} P'}{(\nu xy)P \xrightarrow{\beta} P'} \qquad \textsc{Bet-Inl}\ \frac{P \xrightarrow{x\triangleleft\mathrm{inl} \parallel y\triangleright\mathrm{inl}} P'}{(\nu xy)P \xrightarrow{\beta} (\nu xy)P'} \qquad \textsc{Bet-Inr}\ \frac{P \xrightarrow{x\triangleleft\mathrm{inr} \parallel y\triangleright\mathrm{inr}} P'}{(\nu xy)P \xrightarrow{\beta} (\nu xy)P'}$$

Structural Rules

$$\textsc{Str-Res}\ \frac{P \xrightarrow{\ell} P' \qquad x, y \notin \mathrm{fn}(\ell)}{(\nu xy)P \xrightarrow{\ell} (\nu xy)P'} \qquad \textsc{Str-Par}_1\ \frac{P \xrightarrow{\ell} P' \qquad \mathrm{bn}(\ell) \cap \mathrm{fn}(Q) = \varnothing}{P \parallel Q \xrightarrow{\ell} P' \parallel Q}$$

$$\textsc{Str-Par}_2\ \frac{Q \xrightarrow{\ell} Q' \qquad \mathrm{bn}(\ell) \cap \mathrm{fn}(P) = \varnothing}{P \parallel Q \xrightarrow{\ell} P \parallel Q'} \qquad \textsc{Str-Syn}\ \frac{P \xrightarrow{\ell} P' \qquad Q \xrightarrow{\ell'} Q' \qquad \mathrm{bn}(\ell) \cap \mathrm{bn}(\ell') = \varnothing}{P \parallel Q \xrightarrow{\ell \parallel \ell'} P' \parallel Q'}$$

Figure 7 HCP, label transition semantics.

We define the labelled transition system for HCP as a subsystem of that of Kokke et al. [26], omitting delayed actions. Labels ℓ represent the actions a process can take. Prefixes π are a convenient subset which can be written as prefixes to processes, i. e., $\pi.P$. The label τ represents internal actions. We distinguish two subtypes of internal actions: α represents only the evaluation of links as *renaming*, and β represents only *communication*.

$$\begin{array}{lcl}
\pi & ::= & x[y] \mid x[] \mid x(y) \mid x() \mid x \triangleleft \mathrm{inl} \mid x \triangleleft \mathrm{inr} \\
\ell & := & \pi \mid x{\leftrightarrow}y \mid x \triangleright \mathrm{inl} \mid x \triangleright \mathrm{inr} \mid \tau \mid \alpha \mid \beta
\end{array}$$

We let ℓ_x range over labels on x: $x{\leftrightarrow}y$, $x[y]$, $x[]$, *etc.* Labelled transition $\xrightarrow{\ell}$ is defined in Figure 7. We write $\xrightarrow{\ell}\xrightarrow{\ell'}$ for the composition of $\xrightarrow{\ell}$ and $\xrightarrow{\ell'}$, $\xrightarrow{\ell}_{+}$ for the transitive closure of $\xrightarrow{\ell}$, and $\xrightarrow{\ell}_{\star}$ for the reflexive-transitive closure. We write $\mathrm{bn}(\ell)$ and $\mathrm{fn}(\ell)$ for the bound and free names contained in ℓ, respectively.

The behavioural theory for HCP follows Kokke et al. [26], except that we distinguish two subrelations to bisimilarity, following the subtypes of internal actions.

▶ **Definition 5.1** (Strong bisimilarity). *A relation* $\mathcal{R}$ *on processes is a* strong bisimulation *if* $P \mathrel{\mathcal{R}} Q$ *implies that if* $P \xrightarrow{\ell} P'$, *then* $Q \xrightarrow{\ell} Q'$ *for some* Q' *such that* $P' \mathrel{\mathcal{R}} Q'$. Strong bisimilarity *is the largest relation* $\sim$ *that is a strong bisimulation.*

▶ **Definition 5.2** (Saturated transition). *The* ℓ-saturated transition relation, for $\ell \in \{\alpha, \beta, \tau\}$, is the smallest relation $\Longrightarrow_{\ell}$ such that: $P \overset{\ell}{\Longrightarrow}_{\ell} P$ for all P; and if $P \overset{\ell}{\Longrightarrow}_{\ell} P'$, $P' \xrightarrow{\ell'} Q'$, and $Q' \overset{\ell}{\Longrightarrow}_{\ell} Q$, then $P \overset{\ell'}{\Longrightarrow}_{\ell} Q$. *Saturated transition*, with no qualifier, refers to the τ-saturated transition relation, and is written $\Longrightarrow$.

▶ **Definition 5.3** (Bisimilarity). *A relation $\mathcal{R}$ on processes is an ℓ-bisimulation, for $\ell \in \{\alpha, \beta, \tau\}$, if $P \mathcal{R} Q$ implies that if $P \xRightarrow{\ell'}_{\ell} P'$, then $Q \xRightarrow{\ell'}_{\ell} Q'$ for some Q' such that $P' \mathcal{R} Q'$. The ℓ-bisimilarity relation is the largest relation $\approx_\ell$ that is an ℓ-bisimulation. Bisimilarity, with no qualifier, refers to τ-bisimilarity, and is written $\approx$.*

▶ **Lemma 5.4.** *Structural congruence, strong bisimilarity and the various forms of (weak) bisimilarity are in the expected relation, i. e., $\equiv \subsetneq \sim$, $\sim \subsetneq \approx, \approx_\alpha, \approx_\beta$. Furthermore, bisimilarity is the union of α-bisimilarity and β-bisimilarity, i. e., $\approx = \approx_\alpha \cup \approx_\beta$.*

Translating HGV to HCP. We factor the translation from HGV to HCP into two translations: (1) a translation into HGV$*$, a fine-grain call-by-value [29] variant of HGV, which makes control flow explicit; and (2) a translation from HGV$*$ to HCP.

HGV$*$. We define HGV$*$ as a refinement of HGV in which any non-trivial term must be named by a let binding before being used. While let is syntactic sugar in HGV, it is part of the core language in HGV$*$. Correspondingly, the reduction rule for let follows from the encoding in HGV, i. e.$\textbf{let}\ x = V\ \textbf{in}\ M \longrightarrow_{\mathsf{M}} M\{V/x\}$.

$$
\begin{array}{llcl}
\text{Terms} & L, M, N & ::= & V \mid \textbf{let}\ x = M\ \textbf{in}\ N \mid V\ W \\
 & & \mid & \textbf{let}\ () = V\ \textbf{in}\ M \mid \textbf{let}\ (x, y) = V\ \textbf{in}\ M \\
 & & \mid & \textbf{absurd}\ V \mid \textbf{case}\ V\,\{\textbf{inl}\ x \mapsto M;\ \textbf{inr}\ y \mapsto N\} \\
\text{Values} & V, W & ::= & x \mid K \mid \lambda x.M \mid () \mid (V, W) \mid \textbf{inl}\ V \mid \textbf{inr}\ V \\
\text{Evaluation contexts} & E & ::= & \square \mid \textbf{let}\ x = E\ \textbf{in}\ M
\end{array}
$$

We can *naively* translate HGV to HGV$*$ ($(\!|\cdot|\!)$) by let-binding each subterm in a value position, *e.g.*, $(\!|\textbf{inl}\ M|\!) = \textbf{let}\ z = (\!|M|\!)\ \textbf{in}\ \textbf{inl}\ z$. Such a translation is given in the extended version; standard techniques can be used to avoid administrative redexes [40, 11].

HGV$*$ to HCP. The translation from HGV$*$ to HCP is given in Figure 8. All control flow is encapsulated in values and let-bindings. We define a pair of translations on types, $\lfloor\!\lfloor \cdot \rfloor\!\rfloor$ and $\lceil\!\lceil \cdot \rceil\!\rceil$, such that $\lfloor\!\lfloor T \rfloor\!\rfloor = \lceil\!\lceil T \rceil\!\rceil^\perp$. We extend these translations pointwise to type environments and hyper-environments. We define translations on configurations ($\llbracket \cdot \rrbracket^{\mathsf{c}}_r$), terms ($\llbracket \cdot \rrbracket^{\mathsf{m}}_r$) and values ($\llbracket \cdot \rrbracket^{\mathsf{v}}_r$), where r is a fresh name denoting a special output channel over which the process sends a ping once it has reduced to a value, and then sends the value.

We translate an HGV sequent $\mathcal{G} \parallel \Gamma \vdash \mathcal{C} : T$ as $\llbracket \mathcal{C} \rrbracket^{\mathsf{c}}_r \vdash \lfloor\!\lfloor \mathcal{G} \rfloor\!\rfloor \parallel \lfloor\!\lfloor \Gamma \rfloor\!\rfloor, r : \mathbf{1} \otimes \lfloor\!\lfloor T \rfloor\!\rfloor^\perp$, where Γ is the type environment corresponding to the main thread. The translation of a value $\llbracket V \rrbracket^{\mathsf{v}}_r$ immediately pings the output channel r to announce that it is a value. The translation of a let-binding $\llbracket \textbf{let}\ w = M\ \textbf{in}\ N \rrbracket^{\mathsf{m}}_r$ first evaluates M to a value, which then pings the internal channel x/x' and unblocks the continuation $x.\llbracket N \rrbracket^{\mathsf{m}}_r$.

▶ **Lemma 5.5** (Substitution). *If M is a well-typed term with $w \in \mathrm{fv}(M)$, and V is a well-typed value, then $(\nu w w')(\llbracket M \rrbracket^{\mathsf{m}}_r \parallel \llbracket V \rrbracket^{\mathsf{v}}_{w'}) \approx_\alpha \llbracket M\{V/w\} \rrbracket^{\mathsf{m}}_r$.*

▶ **Theorem 5.6** (Operational Correspondence). *If $\mathcal{C}$ is a well-typed configuration:*

1. *if $\mathcal{C} \longrightarrow \mathcal{C}'$, then $\llbracket \mathcal{C} \rrbracket^{\mathsf{c}}_r \xRightarrow{\beta} \llbracket \mathcal{C}' \rrbracket^{\mathsf{c}}_r$; and*
2. *if $\llbracket \mathcal{C} \rrbracket^{\mathsf{c}}_r \xrightarrow{\beta} P$, then there exists a $\mathcal{C}'$ such that $\mathcal{C} \longrightarrow \mathcal{C}'$ and $P \approx \llbracket \mathcal{C}' \rrbracket^{\mathsf{c}}_r$.*

Translation on types $\boxed{\llfloor T \rrfloor \text{ and } \llceil T \rrceil}$

$$\begin{array}{lll} \llfloor !T.S \rrfloor = \llfloor T \rrfloor^{\perp} \otimes \llfloor S \rrfloor & \llfloor \mathbf{end}_{!} \rrfloor = \mathbf{1} & \llfloor T \rrfloor = \llceil T \rrceil^{\perp}, \\ \llfloor ?T.S \rrfloor = \llfloor T \rrfloor^{\perp} \parr \llfloor S \rrfloor & \llfloor \mathbf{end}_{?} \rrfloor = \perp & \quad \text{if } T \text{ is not a session type} \end{array}$$

$$\begin{array}{lll} \llceil T \times U \rrceil = \llceil T \rrceil \otimes \llceil U \rrceil & \llceil \mathbf{1} \rrceil = \mathbf{1} & \llceil T \multimap U \rrceil = \llceil T \rrceil^{\perp} \parr (\mathbf{1} \otimes \llceil U \rrceil) \\ \llceil T + U \rrceil = \llceil T \rrceil \oplus \llceil U \rrceil & \llceil \mathbf{0} \rrceil = \mathbf{0} & \llceil S \rrceil \qquad\quad = \llfloor S \rrfloor^{\perp} \end{array}$$

Translation on configurations and terms $\boxed{[\![\mathcal{C}]\!]^{\mathsf{c}}_r, [\![V]\!]^{\mathsf{v}}_r, \text{ and } [\![M]\!]^{\mathsf{m}}_r}$

$$\begin{array}{lll} [\![\circ\ M]\!]^{\mathsf{c}}_r = (\nu y y')([\![M]\!]^{\mathsf{m}}_y \parallel y'.y'[].\mathbf{0}) & [\![(\nu x x')\mathcal{C}]\!]^{\mathsf{c}}_r = (\nu x x')[\![\mathcal{C}]\!]^{\mathsf{c}}_r & [\![x \overset{z}{\leftrightarrow} y]\!]^{\mathsf{c}}_r = \bar{z}.z().x{\leftrightarrow}y \\ [\![\bullet\ M]\!]^{\mathsf{c}}_r = [\![M]\!]^{\mathsf{m}}_r & [\![\mathcal{C} \parallel \mathcal{D}]\!]^{\mathsf{c}}_r \ = [\![\mathcal{C}]\!]^{\mathsf{c}}_r \parallel [\![\mathcal{D}]\!]^{\mathsf{c}}_r & \end{array}$$

$$\begin{array}{lll} [\![x]\!]^{\mathsf{v}}_r \qquad = r{\leftrightarrow}x & [\![()]\!]^{\mathsf{v}}_r \qquad = r[].\mathbf{0} & [\![\mathbf{inl}\ V]\!]^{\mathsf{v}}_r = r \triangleleft \mathrm{inl}.[\![V]\!]^{\mathsf{v}}_r \\ [\![\lambda x.M]\!]^{\mathsf{v}}_r = r(x).[\![M]\!]^{\mathsf{m}}_r & [\![(V, W)]\!]^{\mathsf{v}}_r = r[x].([\![V]\!]^{\mathsf{v}}_x \parallel [\![W]\!]^{\mathsf{v}}_r) & [\![\mathbf{inr}\ V]\!]^{\mathsf{v}}_r = r \triangleleft \mathrm{inr}.[\![V]\!]^{\mathsf{v}}_r \end{array}$$

$$\begin{array}{ll} [\![V\ W]\!]^{\mathsf{m}}_r & = (\nu x x')(\nu y y')(y\langle x\rangle.r{\leftrightarrow}y \parallel [\![V]\!]^{\mathsf{v}}_{y'} \parallel [\![W]\!]^{\mathsf{v}}_{x'}) \\ [\![\mathbf{let}\ () = V\ \mathbf{in}\ M]\!]^{\mathsf{m}}_r & = (\nu x x')(x().[\![M]\!]^{\mathsf{m}}_r \parallel [\![V]\!]^{\mathsf{v}}_{x'}) \\ [\![\mathbf{let}\ (x, y) = V\ \mathbf{in}\ M]\!]^{\mathsf{m}}_r & = (\nu y y')(y(x).[\![M]\!]^{\mathsf{m}}_r \parallel [\![V]\!]^{\mathsf{v}}_{y'}) \\ [\![\mathbf{case}\ V\ \{\mathbf{inl}\ x \mapsto M;\ \mathbf{inr}\ y \mapsto N\}]\!]^{\mathsf{m}}_r & = (\nu x x')(x \triangleright \{\mathrm{inl} : [\![M]\!]^{\mathsf{m}}_r; \mathrm{inr} : [\![N\{x/y\}]\!]^{\mathsf{m}}_r\} \parallel [\![V]\!]^{\mathsf{v}}_{x'}) \\ [\![\mathbf{absurd}\ V]\!]^{\mathsf{m}}_r & = (\nu x x')(x \triangleright \{\} \parallel [\![V]\!]^{\mathsf{v}}_{x'}) \\ [\![\mathbf{let}\ x = M\ \mathbf{in}\ N]\!]^{\mathsf{m}}_r & = (\nu x x')(x.[\![N]\!]^{\mathsf{m}}_r \parallel [\![M]\!]^{\mathsf{m}}_{x'}) \\ [\![V]\!]^{\mathsf{m}}_r & = \bar{r}.[\![V]\!]^{\mathsf{v}}_r \end{array}$$

$$\begin{array}{lll} [\![\mathbf{link}]\!]^{\mathsf{v}}_r = r(y).y(x).\bar{r}.r().x{\leftrightarrow}y & [\![\mathbf{send}]\!]^{\mathsf{v}}_r = r(y).y(x).y\langle x\rangle.\bar{r}.r{\leftrightarrow}y & [\![\mathbf{wait}]\!]^{\mathsf{v}}_r = r(x).x().\bar{r}.r[].\mathbf{0} \\ [\![\mathbf{fork}]\!]^{\mathsf{v}}_r = r(x).\bar{r}.x\langle r\rangle.x.x[].\mathbf{0} & [\![\mathbf{recv}]\!]^{\mathsf{v}}_r = r(x).x(y).\bar{r}.r\langle y\rangle.r{\leftrightarrow}x & \end{array}$$

Figure 8 Translation from HGV$*$ to HCP.

Translating HCP to HGV. We cannot translate HCP processes to HGV terms directly: HGV's term language only supports **fork** (see the extended version for further discussion), so there is no way to translate an individual name restriction or parallel composition. However, we can still translate HCP into HGV via the composition of known translations.

HCP into CP We must first reunite each parallel composition with its corresponding name restriction, i. e., translate to CP using the *disentanglement* translation shown by Kokke *et al.* [27, Lemma 4.7]. The result is a collection of independent CP processes.

CP into GV Next, we can translate each CP process into a GV configuration using (a variant of) Lindley and Morris' translation [30, Figure 8].

GV into HGV Finally, we can use our embedding of GV into HGV (Theorem 4.3) to obtain a collection of well-typed HGV configurations, which can be composed using TC-Par to result in a single well-typed HGV configuration.

The translation from HCP into CP and the embedding of GV into HGV preserve and reflect reduction. However, Lindley and Morris's original translation from CP to GV preserves but does not reflect reduction due to an asynchronous encoding of choice. By adapting their translation to use a synchronous encoding of choice (Section 3), we obtain a translation from CP to GV that both preserves and reflects reduction. Thus, composing all three translations together we obtain a translation from HCP to HGV that preserves and reflects reduction.

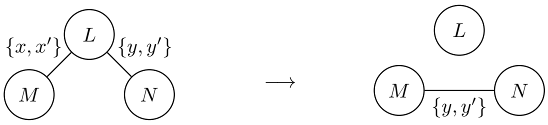
6 Extensions

In this section, we outline three extensions to HGV that exploit generalising the tree structure of processes to a forest structure. Full details are given in the extended version. These extensions are of particular interest since HGV already supports a core aspect of forest structure, enabling its full utilisation merely through the addition of a structural rule. In contrast, to extend GV with forest structure one must distinguish two distinct introduction rules for parallel composition [30]. Other extensions to GV such as shared channels [30], polymorphism [32], and recursive session types [31] adapt to HGV almost unchanged.

From trees to forests. The TC-Mix structural rule allows two type environments Γ_1, Γ_2 to be split by a hyper-environment separator *without* a channel connecting them. Mix [17] may be interpreted as concurrency *without* communication [30, 3].

$$\textsc{TC-Mix}\ \frac{\mathcal{G} \parallel \Gamma_1 \parallel \Gamma_2 \vdash \mathcal{C} : T}{\mathcal{G} \parallel \Gamma_1, \Gamma_2 \vdash \mathcal{C} : T}$$

A simpler link. Consider threads $L = F[\textbf{link}\ (x, y)]$, M, N, where L connects to M by x and to N by y.



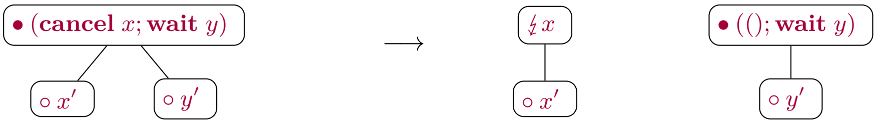
The result of link reduction has forest structure. Well-typed closed programs in both GV and HGV must *always* maintain tree structure. Different versions of GV do so in various unsatisfactory ways: one is pre-emptive blocking [30], which breaks confluence; another is two stage linking (Figure 4), which defers forwarding via a special link thread [31]. With TC-Mix, we can adjust the type schema for **link** to $(S \times \overline{S}) \multimap \mathbf{1}$ and use the following rule.

$$\textsc{E-Link-Mix} \qquad (\nu x x')(\mathcal{F}[\textbf{link}\ (x, y)] \parallel \phi N) \longrightarrow \mathcal{F}[()] \parallel \phi N\{y/x'\}$$

This formulation enables immediate substitution, maximimising concurrency.

Exceptions. In order to support exceptions in the presence of linear endpoints [14, 35] we must have a way of *cancelling* an endpoint ($\textbf{cancel} : S \multimap \mathbf{1}$). Cancellation generates a special *zapper thread* ($\lightning x$) which severs a tree topology into a forest as in the following example.

$$(\nu x x')(\nu y y')(\circ x' \parallel \circ y' \parallel \bullet(\textbf{cancel}\ x; \textbf{wait}\ y)) \qquad (\nu x x')(\nu y y')(\circ x' \parallel \circ y' \parallel \lightning x \parallel \bullet((); \textbf{wait}\ y)$$



7 Related work

Session Types and Functional Languages. HGV traces its origins to a line of work initiated by Gay and collaborators [15, 48, 50, 16]. This family of calculi builds session types directly into a lambda calculus. Toninho *et al.* [47] take an alternative approach, stratifying their system into a session-typed process calculus and a separate functional calculus. There are many pragmatic embeddings of session type systems in existing functional programming languages [36, 41, 43, 21, 38, 24]. A detailed survey is given by Orchard & Yoshida [37].

Propositions as Sessions. When Girard introduced linear logic [17] he suggested a connection with concurrency. Abramsky [1] and Bellin and Scott [5] give embeddings of linear logic proofs in π-calculus, where cut reduction is simulated by π-calculus reduction. Both embeddings interpret tensor as parallel composition. The correspondence with π-calculus is not tight in that these systems allow independent prefixes to be reordered. Caires and Pfenning [7] give a propositions as types correspondence between dual intuitionistic linear logic and a session-typed π-calculus called πDILL. They interpret tensor as output. The correspondence with π-calculus is tight in that independent prefixes may not be reordered. With CP [51], Wadler adapts πDILL to classical linear logic. Aschieri and Genco [2] give an interpretation of classical multiplicative linear logic as concurrent functional programs. They interpret $\parr$ as parallel composition, and the connection to session types is less direct.

Priority-based Calculi. Systems such as πDILL, CP, and GV (and indeed HCP and HGV) ensure deadlock freedom by exploiting the type system to statically impose a tree structure on the communication topology – there can be at most one communication channel between any two processes. Another line of work explores a more liberal approach to deadlock freedom enabling some cyclic communication topologies, where deadlock freedom is guaranteed via *priorities*, which impose an order on actions. Priorites were introduced by Kobayashi and Padovani [23, 39] and adopted by Dardha and Gay [12] in Priority CP (PCP) and Kokke and Dardha in Priority GV (PGV) [25].

8 Conclusion and future work

HGV exploits hypersequents to resolve fundamental modularity issues with GV. As a consequence, we have obtained a tight operational correspondence between HGV and HCP. HGV is a modular and extensible core calculus for functional programming with *binary* session types. In future we intend to further exploit hypersequents in order to develop a modular and extensible core calculus for functional programming with *multiparty* session types. We would then hope to exhibit a similarly tight operational correspondence between this functional calculus and a multiparty variant of CP [9].

References

1 Samson Abramsky. Proofs as processes. *Theoretical Computer Science*, 135(1):5–9, 1994.

2 Federico Aschieri and Francesco A. Genco. Par means parallel: multiplicative linear logic proofs as concurrent functional programs. *Proc. ACM Program. Lang.*, 4(POPL):18:1–18:28, 2020.

3 Robert Atkey, Sam Lindley, and J. Garrett Morris. Conflation confers concurrency. In *A List of Successes That Can Change the World*, volume 9600 of *Lecture Notes in Computer Science*, pages 32–55. Springer, 2016.

4 Arnon Avron. Hypersequents, logical consequence and intermediate logics for concurrency. *Ann. Math. Artif. Intell.*, 4:225–248, 1991.

5 Gianluigi Bellin and Philip J. Scott. On the pi-calculus and linear logic. *Theoretical Computer Science*, 135(1):11–65, 1994.

6 Michele Boreale. On the expressiveness of internal mobility in name-passing calculi. *Theoretical Computer Science*, 195(2):205–226, March 1998. `doi:10.1016/s0304-3975(97)00220-x`.

7 Luìs Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In *Proc. of CONCUR*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010.

8 Marco Carbone, Ornela Dardha, and Fabrizio Montesi. Progress as compositional lock-freedom. In *Proc. of COORDINATION*, volume 8459 of *Lecture Notes in Computer Science*, pages 49–64. Springer, 2014. doi:10.1007/978-3-662-43376-8_4.

9 Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schürmann, and Philip Wadler. Coherence generalises duality: A logical explanation of multiparty session types. In *CONCUR*, volume 59 of *LIPIcs*, pages 33:1–33:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

10 Marco Carbone, Fabrizio Montesi, and Carsten Schürmann. Choreographies, logically. *Distributed Comput.*, 31(1):51–67, 2018.

11 Olivier Danvy, Kevin Millikin, and Lasse R. Nielsen. On one-pass CPS transformations. *J. Funct. Program.*, 17(6):793–812, 2007.

12 Ornela Dardha and Simon J. Gay. A new linear logic for deadlock-free session-typed processes. In *Proc. of FoSSaCS*, volume 10803 of *LNCS*, pages 91–109. Springer, 2018.

13 Ornela Dardha, Elena Giachino, and Davide Sangiorgi. Session types revisited. *Inf. Comput.*, 256:253–286, 2017.

14 Simon Fowler, Sam Lindley, J. Garrett Morris, and Sára Decova. Exceptional asynchronous session types: session types without tiers. *Proc. ACM Program. Lang.*, 3(POPL):28:1–28:29, 2019.

15 Simon J. Gay and Rajagopal Nagarajan. Intensional and extensional semantics of dataflow programs. *Formal Aspects of Computing*, 15(4):299–318, 2003.

16 Simon J. Gay and Vasco T. Vasconcelos. Linear type theory for asynchronous session types. *Journal of Functional Programming*, 20(1):19–50, 2010.

17 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.

18 Kohei Honda. Types for dyadic interaction. In *CONCUR*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993.

19 Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language primitives and type discipline for structured communication-based programming. In *Proc. of ESOP*, volume 1381 of *LNCS*, pages 122–138. Springer, 1998.

20 Atsushi Igarashi, Peter Thiemann, Yuya Tsuda, Vasco T. Vasconcelos, and Philip Wadler. Gradual session types. *J. Funct. Program.*, 29:e17, 2019.

21 Keigo Imai, Shoji Yuen, and Kiyoshi Agusa. Session type inference in Haskell. In *Proc. of PLACES*, volume 69 of *EPTCS*, pages 74–91, 2010. doi:10.4204/EPTCS.69.6.

22 Naoki Kobayashi. Type systems for concurrent programs. In Bernhard K. Aichernig and Tom Maibaum, editors, *Formal Methods at the Crossroads. From Panacea to Foundational Support: 10th Anniversary Colloquium of UNU/IIST, the International Institute for Software Technology of The United Nations University, Lisbon, Portugal, March 18-20, 2002. Revised Papers*, pages 439–453. Springer Berlin Heidelberg, 2003. doi:10.1007/978-3-540-40007-3_26.

23 Naoki Kobayashi. A new type system for deadlock-free processes. In *Proc. of CONCUR*, volume 4137 of *LNCS*, pages 233–247. Springer, 2006.

24 Wen Kokke and Ornela Dardha. Deadlock-free session types in linear Haskell. *CoRR*, abs/2103.14481, 2021. Accepted for publication at the Haskell Symposium 2021. arXiv:2103.14481.

25 Wen Kokke and Ornela Dardha. Prioritise the best variation. In *FORTE*, volume 12719 of *Lecture Notes in Computer Science*, pages 100–119. Springer, 2021.

26 Wen Kokke, Fabrizio Montesi, and Marco Peressotti. Better late than never: A fully-abstract semantics for classical processes. *PACMPL*, 3(POPL), 2019.

27 Wen Kokke, Fabrizio Montesi, and Marco Peressotti. Taking linear logic apart. In Thomas Ehrhard, Maribel Fernández, Valeria de Paiva, and Lorenzo Tortora de Falco, editors, *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications, Oxford, UK, 7-8 July 2018*, volume 292 of *Electronic Proceedings in Theoretical Computer Science*, pages 90–103. Open Publishing Association, 2019.

28 Jean-Jacques Lévy and Luc Maranget. Explicit substitutions and programming languages. In *Foundations of Software Technology and Theoretical Computer Science, 1999*, volume 1738 of *LNCS*. Springer, 1999. doi:10.1007/3-540-46691-6_14.

29 Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Inf. Comput.*, 185(2):182–210, 2003.

30 Sam Lindley and J. Garrett Morris. A semantics for propositions as sessions. In Jan Vitek, editor, *Programming Languages and Systems*, pages 560–584. Springer Berlin Heidelberg, 2015.

31 Sam Lindley and J. Garrett Morris. Talking bananas: Structural recursion for session types. *SIGPLAN Not.*, 51(9):434–447, 2016. doi:10.1145/3022670.2951921.

32 Sam Lindley and J. Garrett Morris. Lightweight functional session types. In Simon Gay and Antonio Ravara, editors, *Behavioural Types: from Theory to Tools*, chapter 12, pages 265–286. River publishers, 2017.

33 Fabrizio Montesi. *Choreographic Programming.* PhD thesis, IT University of Copenhagen, 2013.

34 Fabrizio Montesi and Marco Peressotti. Classical transitions. *CoRR*, abs/1803.01049, 2018. arXiv:1803.01049.

35 Dimitris Mostrous and Vasco T. Vasconcelos. Affine sessions. *Log. Methods Comput. Sci.*, 14(4), 2018.

36 Matthias Neubauer and Peter Thiemann. An implementation of session types. In *Proc. of PADL*, volume 3057 of *Lecture Notes in Computer Science*, pages 56–70. Springer, 2004. doi:10.1007/978-3-540-24836-1_5.

37 Dominic Orchard and Nobuko Yoshida. Session types with linearity in Haskell. *Behavioural Types: from Theory to Tools*, page 219, 2017.

38 Dominic A. Orchard and Nobuko Yoshida. Effects as sessions, sessions as effects. In *Proc. of POPL*, pages 568–581. ACM, 2016. doi:10.1145/2837614.2837634.

39 Luca Padovani. Deadlock and Lock Freedom in the Linear π-Calculus. In *Proc. of CSL-LICS*, pages 72:1–72:10. ACM, 2014.

40 Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.

41 Riccardo Pucella and Jesse A. Tov. Haskell session types with (almost) no class. In *Proc. of Haskell*. ACM, 2008. doi:10.1145/1411286.1411290.

42 John C. Reynolds. The meaning of types—from intrinsic to extrinsic semantics. Technical Report RS-00-32, BRICS, 2000.

43 Matthew Sackman and Susan Eisenbach. Session types in Haskell: Updating message passing for the 21st century. Unpublished manuscript, 2008.

44 Davide Sangiorgi. π-calculus, internal mobility, and agent-passing calculi. *Theoretical Computer Science*, 167(1-2):235–274, 1996. doi:10.1016/0304-3975(96)00075-8.

45 Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An interaction-based language and its typing system. In *Proc. of PARLE*, volume 817 of *LNCS*, pages 398–413. Springer, 1994.

46 Peter Thiemann and Vasco T. Vasconcelos. Label-dependent session types. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–29, 2020.

47 Bernardo Toninho, Luís Caires, and Frank Pfenning. Higher-order processes, functions, and sessions: A monadic integration. In *ESOP*, volume 7792 of *Lecture Notes in Computer Science*, pages 350–369. Springer, 2013.

48 Vasco Vasconcelos, Antonio Ravara, and Simon J. Gay. Session types for functional multithreading. In *CONCUR*, volume 3170 of *LNCS*, pages 497–511. Springer, 2004.

49 Vasco T. Vasconcelos. Fundamentals of session types. *Inf. Comput.*, 217:52–70, 2012.

50 Vasco Thudichum Vasconcelos, Simon J. Gay, and Antonio Ravara. Type checking a multithreaded functional language with session types. *Theor. Comput. Sci.*, 368(1-2):64–87, 2006.

51 Philip Wadler. Propositions as sessions. *Journal of Functional Programming*, 24(2-3):384–418, 2014.