30th International Conference on Concurrency Theory

CONCUR 2019, August 27–30, 2019,
Amsterdam, the Netherlands

Edited by

Wan Fokkink
Rob van Glabbeek

LIPICS

Editors

Wan Fokkink
Vrije Universiteit Amsterdam, the Netherlands
w.j.fokkink@vu.nl

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

ACM Classification 2012
Theory of computation → Concurrency

ISBN 978-3-95977-121-4

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at https://www.dagstuhl.de/dagpub/978-3-95977-121-4.

Publication date
August, 2019

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.2019.0

ISBN 978-3-95977-121-4 **ISSN 1868-8969** **https://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

ISSN 1868-8969

https://www.dagstuhl.de/lipics

Contents

Preface
Wan Fokkink and Rob van Glabbeek 0:ix

Invited Contributions

Safety Verification for Deep Neural Networks with Provable Guarantees
Marta Z. Kwiatkowska 1:1–1:5

Synthesis of Safe, Optimal and Compact Strategies for Stochastic Hybrid Games
Kim G. Larsen 2:1–2:5

Program Invariants
Joël Ouaknine 3:1–3:1

Concurrent Algorithms and Data Structures for Model Checking
Jaco van de Pol 4:1–4:1

Markov Decision Processes

Of Cores: A Partial-Exploration Framework for Markov Decision Processes
Jan Křetínský and Tobias Meggendorfer 5:1–5:17

Combinations of Qualitative Winning for Stochastic Parity Games
Krishnendu Chatterjee and Nir Piterman 6:1–6:17

Near-Linear Time Algorithms for Streett Objectives in Graphs and MDPs
Krishnendu Chatterjee, Wolfgang Dvořák, Monika Henzinger, and Alexander Svozil 7:1–7:16

Life Is Random, Time Is Not: Markov Decision Processes with Window Objectives
Thomas Brihaye, Florent Delgrange, Youssouf Oualhadj, and Mickael Randour . . . 8:1–8:18

Probabilistic systems

Computing Probabilistic Bisimilarity Distances for Probabilistic Automata
Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, Radu Mardare, Qiyi Tang, and Franck van Breugel 9:1–9:17

Asymmetric Distances for Approximate Differential Privacy
Dmitry Chistikov, Andrzej S. Murawski, and David Purser 10:1–10:17

Event Structures for Mixed Choice
Marc de Visme 11:1–11:16

Reachability

Verification of Flat FIFO Systems
Alain Finkel and M. Praveen 12:1–12:17

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

The Complexity of Subgame Perfect Equilibria in Quantitative Reachability Games
Thomas Brihaye, Véronique Bruyère, Aline Goeminne, Jean-François Raskin, and Marie van den Bogaard 13:1–13:16

On the Complexity of Reachability in Parametric Markov Decision Processes
Tobias Winkler, Sebastian Junges, Guillermo A. Pérez, and Joost-Pieter Katoen . 14:1–14:17

Timed Basic Parallel Processes
Lorenzo Clemente, Piotr Hofman, and Patrick Totzke 15:1–15:16

Automata

Revisiting Local Time Semantics for Networks of Timed Automata
R. Govind, Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz 16:1–16:15

Approximate Learning of Limit-Average Automata
Jakub Michaliszyn and Jan Otop 17:1–17:16

Alternating Weak Automata from Universal Trees
Laure Daviaud, Marcin Jurdziński, and Karoliina Lehtinen 18:1–18:14

Good for Games Automata: From Nondeterminism to Alternation
Udi Boker and Karoliina Lehtinen 19:1–19:16

Games

Determinacy in Discrete-Bidding Infinite-Duration Games
Milad Aghajohari, Guy Avni, and Thomas A. Henzinger 20:1–20:17

Energy Mean-Payoff Games
Véronique Bruyère, Quentin Hautem, Mickael Randour, and Jean-François Raskin 21:1–21:17

Equilibrium Design for Concurrent Games
Julian Gutierrez, Muhammad Najib, Giuseppe Perelli, and Michael Wooldridge ... 22:1–22:16

Partial Order Reduction for Reachability Games
Frederik Meyer Bønneland, Peter Gjøl Jensen, Kim G. Larsen, Marco Muñiz, and Jiří Srba 23:1–23:15

Synthesis

Synthesis of Data Word Transducers
Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier 24:1–24:15

Register-Bounded Synthesis
Ayrat Khalimov and Orna Kupferman 25:1–25:16

Translating Asynchronous Games for Distributed Synthesis
Raven Beutner, Bernd Finkbeiner, and Jesko Hecking-Harbusch 26:1–26:16

VASS and concurrent programming

Long-Run Average Behavior of Vector Addition Systems with States
Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop 27:1–27:16

Reachability for Bounded Branching VASS
Filip Mazowiecki and Michał Pilipczuk 28:1–28:13

Reasoning About Distributed Knowledge of Groups with Infinitely Many Agents
Michell Guzmán, Sophia Knight, Santiago Quintero, Sergio Ramírez, Camilo Rueda, and Frank Valencia 29:1–29:15

Robustness Against Transactional Causal Consistency
Sidi Mohamed Beillahi, Ahmed Bouajjani, and Constantin Enea 30:1–30:18

Broadcast and fault-tolerance

Expressive Power of Broadcast Consensus Protocols
Michael Blondin, Javier Esparza, and Stefan Jaax 31:1–31:16

Reconfiguration and Message Losses in Parameterized Broadcast Networks
Nathalie Bertrand, Patricia Bouyer, and Anirban Majumdar 32:1–32:15

Verification of Randomized Consensus Algorithms Under Round-Rigid Adversaries
Nathalie Bertrand, Igor Konnov, Marijana Lazić, and Josef Widder 33:1–33:15

A Sound Foundation for the Topological Approach to Task Solvability
Jérémy Ledent and Samuel Mimram 34:1–34:15

Coalgebra and model checking

Game-Based Local Model Checking for the Coalgebraic μ-Calculus
Daniel Hausmann and Lutz Schröder 35:1–35:16

Graded Monads and Graded Logics for the Linear Time – Branching Time Spectrum
Ulrich Dorsch, Stefan Milius, and Lutz Schröder 36:1–36:16

Bialgebraic Semantics for String Diagrams
Filippo Bonchi, Robin Piedeleu, Paweł Sobocinski, and Fabio Zanasi 37:1–37:17

Session types and Kleene algebra

A Sound Algorithm for Asynchronous Session Subtyping
Mario Bravetti, Marco Carbone, Julien Lange, Nobuko Yoshida, and Gianluigi Zavattaro 38:1–38:16

Domain-Aware Session Types
Luís Caires, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho 39:1–39:17

Reordering Derivatives of Trace Closures of Regular Languages
Hendrik Maarand and Tarmo Uustalu 40:1–40:16

Kleene Algebra with Observations
Tobias Kappé, Paul Brunet, Jurriaan Rot, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi 41:1–41:16

Preface

This volume contains the proceedings of the 30st Conference on Concurrency Theory, which was held in Amsterdam, The Netherlands, on August 27–30, 2019. CONCUR 2019 was organized by CWI. At its 30th anniversary the CONCUR conference series has returned to its place of birth, as the first two instalments, in 1990 and 1991, took place in Amsterdam.

CONCUR is a forum for the development and dissemination of leading research in concurrency theory and its applications. Its aim is to bring together researchers, developers, and students to exchange and discuss latest theoretical developments and learn about challenging practical problems. CONCUR is the reference annual event for researchers in the field.

The principal topics include basic models of concurrency such as abstract machines, domain-theoretic models, game-theoretic models, process algebras, graph transformation systems, Petri nets, hybrid systems, mobile and collaborative systems, probabilistic systems, real-time systems, biology-inspired systems, and synchronous systems; logics for concurrency such as modal logics, probabilistic and stochastic logics, temporal logics, and resource logics; verification and analysis techniques for concurrent systems such as abstract interpretation, atomicity checking, model checking, race detection, pre-order and equivalence checking, run-time verification, state-space exploration, static analysis, synthesis, testing, theorem proving, type systems, and security analysis; distributed algorithms and data structures: design, analysis, complexity, correctness, fault tolerance, reliability, availability, consistency, self-organisation, self-stabilisation, protocols. Also the theoretical foundations of more applied topics like architectures, execution environments, and software development for concurrent systems such as geo-replicated systems, communication networks, multiprocessor and multi-core architectures, shared and transactional memory, resource management and awareness, compilers and tools for concurrent programming, programming models such as component-based, object- and service-oriented can be found at CONCUR.

This edition of the conference attracted 93 full paper submissions. We thank the authors for their interest in CONCUR 2018. After careful reviewing and discussions, the Program Committee selected 37 papers for presentation at the conference. Each submission was reviewed by at least three reviewers who wrote detailed evaluations and gave insightful comments. We warmly thank the members of the Program Committee and the additional reviewers for their excellent work, including the constructive discussions. The full list of reviewers is available as part of these proceedings.

The conference programme was greatly enriched by the invited presentations by Marta Kwiatkowska (University of Oxford, UK), Kim G. Larsen (Aalborg University, Denmark), Joël Ouaknine (Max Planck Institute for Software Systems, Germany) and Jaco van de Pol (Aarhus University, Denmark). The abstracts of their talks are available as part of these proceedings.

This year, the conference was jointly organised with the 17th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 2019) and the 24th International Conference on Formal Methods for Industrial Critical Systems (FMICS 2019). Additionally, the event was enriched by six satellite events: the Combined 26th International Workshop on Expressiveness in Concurrency and 16th Workshop on Structural Operational Semantics (EXPRESS/SOS 2019), the 8th IFIP WG 1.8 Workshop on Trends in Concurrency Theory (TRENDS 2019), the 9th Young Researchers Workshop on Concurrency Theory (YR-CONCUR 2019), the 4th International Workshop on Timing Performance Engineering for Safety Critical Systems (TIPS 2019), the 3rd International Workshop on Methods and Tools for Distributed Hybrid Systems (DHS 2019), and the 2nd International Workshop on Recent Advances in Concurrency and Logic (RADICAL 2019).

As usual, the CONCUR proceedings are available for open access via LIPIcs. We thank the authors and the participants for making this year's CONCUR a success.

Wan Fokkink (Vrije Universiteit Amsterdam, The Netherlands)
Rob van Glabbeek (Data61, CSIRO, Sydney, Australia)

Committees

Programme Committee

Christel Baier

Jiri Barnat

Benedikt Bollig

Borzoo Bonakdarpour

Ilaria Castellani

Taolue Chen

Rance Cleaveland

Yuxin Deng

Josée Desharnais

Adrian Francalanza

Wan Fokkink (co-chair)

Ansgar Fehnker

David de Frutos-Escrig

Yuxi Fu

Rob van Glabbeek (co-chair)

Alexey Gotsman

Radu Grosu

Ichiro Hasuo

Marieke Huisman

Barbara König

Gerald Lüttgen

Bas Luttik

Anca Muscholl

Uwe Nestmann

Jun Pang

Jean-François Raskin

Grigore Rosu

Jiri Srba

Simone Tini

Frank Valencia

James Worrell

Gianluigi Zavattaro

Steering Committee

Javier Esparza

Pedro D'Argenio

Wan Fokkink

Joost-Pieter Katoen

Catuscia Palamidessi

Davide Sangiorgi

Jiri Srba

General Chair

Jos Baeten

Workshop Chair

Bas Luttik

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Samy Abbes
Luca Aceto
Antonis Achilleos
C. Aiswarya
S. Akshay
Étienne André
Alasdair Armstrong
Paolo Baldan
Jaroslav Bendík
Nikola Beneš
Giovanni Bernardi
Raphaël Berthon
Laura Bocchi
Filippo Bonchi
Patricia Bouyer
Luboš Brim
Paul Brunet
Frederik M. Bønneland
Marco Carbone
Valentina Castiglioni
Ivana Černá
Andrea Cerone
Marek Chalupa
Minas Charalambides
Xiaohong Chen
Corina Cirstea
Lorenzo Clemente
Thomas Colcombet
Silvia Crafa
Cas Cremers
Sandeep Dasgupta
Marc de Visme
Giorgio Delzanno
Laurent Doyen
Jannik Dreier
Jérémy Dubut
Richard Eggert
Nathanael Fijalkov
János Flesch
Simon Fowler
Dan Frumin
Hongfei Fu
Ignacio Fábregas
Maurizio Gabbrielli
Pierre Ganty
Patrick Gardy
Gilles Geeraerts
Raffaella Gentilini
Paola Giannini
Thomas Given-Wilson
Shibashis Guha
Julian Gutierrez
Peter Habermehl
Ernst Moritz Hahn
Daniel Hirschkoff
Ross Horne
Justin Hsu
Mingzhang Huang
Md. Ariful Islam
Guilhem Jaber
Stefan Jaksic
Wojtek Jamroga
Christian Johansen
Sebastiaan Joosten
Vincent Jugé
Marcin Jurdzinski
Tobias Kappé
Isabella Kaufmann
Bartek Klin
Alexander Knapp
Sophia Knight
Siddharth Krishna
Jean Krivine
Satoshi Kura
Ori Lahav
Yngve Lamo
Ivan Lanese
Julien Lange
Ruggero Lanotte
Sophie Lathouwers
Henrich Lauko
Stéphane Le Roux
Mathieu Lehaut
Karoliina Lehtinen
Huan Long
Florian Lorber
Michele Loreti
Anna Lukina
Nicolas Markey
Johannes Marti

Richard Mayr
Michael Mendler
Philipp J. Meyer
Jan Midtgaard
Lukasz Mikulski
Marius Mikučionis
Arthur Milchior
Andrzej Mizera
Joshua Moerman
Jean-Francois Monin
Benjamin Monmege
Marco Muniz
David Müller
Thomas Neele
Wytse Oortwijn
Youssouf Oualhadj
Luca Padovani
Miguel Palomino
Samuel Pastva
Soumya Paul
Adriano Peron
Kirstin Peters
Anna Philippou
Jakob Piribauer
Nir Piterman
Chris Poskitt
Damien Pous
Vinayak Prabhu
M. Praveen
Tobias Prehn
Guillermo Pérez
Jorge A. Pérez
Karin Quaas
Tim Quatmann
Yasmeen Rafiq
Sergio Rajsbaum
Steven Ramsay
Mickael Randour
Vishal Jagannath Ravi
Christina Rickmann
Chloe Rispal
Camilo Rueda
Claudio Sacerdoti Coen
Mohsen Safari
Prakash Saivasan
Matteo Sammartino
Arnaud Sangnier
Zdeněk Sawa
Sylvain Schmitz
Olivier Serre
Pawel Sobocinski
Ana Sokolova
Marcelo Sousa
Gheorghe Stefanescu
Nathalie Sznajder
Vasco T. Vasconcelos
Toru Takisaka
Max Tschaikowski
Takeshi Tsukada
Andrea Turrini
Irek Ulidowski
Franck van Breugel
Marie Van Den Bogaard
Ingo van Duijn
Georg Weissenbacher
Tim Willemse
Sascha Wunderlich
Ming Xu
Xian Xu
Qizhe Yang
Eugene Yip
Margherita Zorzi

Safety Verification for Deep Neural Networks with Provable Guarantees

Marta Z. Kwiatkowska
Department of Computer Science, University of Oxford, UK
http://www.cs.ox.ac.uk/marta.kwiatkowska/
marta.kwiatkowska@cs.ox.ac.uk

Abstract

Computing systems are becoming ever more complex, increasingly often incorporating deep learning components. Since deep learning is unstable with respect to adversarial perturbations, there is a need for rigorous software development methodologies that encompass machine learning. This paper describes progress with developing automated verification techniques for deep neural networks to ensure safety and robustness of their decisions with respect to input perturbations. This includes novel algorithms based on feature-guided search, games, global optimisation and Bayesian methods.

2012 ACM Subject Classification Theory of computation → Logic and verification; Computing methodologies → Neural networks

Keywords and phrases Neural networks, robustness, formal verification, Bayesian neural networks

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.1

Category Invited Paper

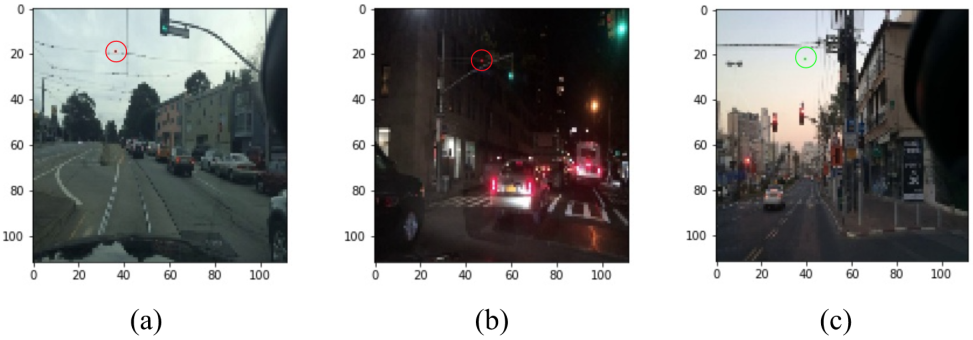
Funding *Marta Z. Kwiatkowska*: This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 834115) and the EPSRC Programme Grant on Mobile Autonomy (EP/M019918/1).

1 Introduction

Computing devices have become ubiquitous and ever present in our lives: smartphones help us stay in touch with family and friends, GPS-enabled apps offer directions literally at our fingertips, and voice-controlled assistants are now able to execute simple commands. Artificial Intelligence is making great strides, promising many more exciting applications with an increased level of autonomy, from wearable medical devices to robotic care assistants and self-driving cars.

Deep learning, in particular, is revolutionising AI. Deep neural networks (DNNs) have been developed for a variety of tasks, including computer vision, face recognition, malware detection, speech recognition and text analysis. While the accuracy of neural networks has greatly improved, they are susceptible to *adversarial examples* [17, 1]. An adversarial example is an input which, though initially classified correctly, is misclassified after a minor, perhaps imperceptible, perturbation. Figure 1 from [19] shows an image of a traffic light correctly classified by a convolutional neural network, which is then misclassified after changing only a few pixels. This illustrative example, though somewhat artificial, since in practice the controller would rely on additional sensor input when making a decision, highlights the need for appropriate mechanisms and frameworks to prevent the occurrence of similar issues during deployment.

Clearly, the excitement surrounding the potential of AI and autonomous computing technologies is well placed. Autonomous devices make decisions on their own and on users' behalf, powered by software that today often incorporates machine learning components. Since autonomous device technologies are increasingly often incorporated within safety-



(a) (b) (c)

Figure 1 from [19]. Adversarial examples generated on Nexar challenge data (dashboard camera images). (a) Green light classified as red with confidence 56% after one pixel change. (b) Green light classified as red with confidence 76% after one pixel change. (c) Red light classified as green with 90% confidence after one pixel change.

critical applications, they must trustworthy. However, software faults can have disastrous consequences, potentially resulting in fatalities. Given the complexity of the scenarios and uncertainty in the environment, it is important to ensure that software incorporating machine learning components is robust and safe.

2 Overview of progress in automated verification for neural networks

Robustness (or resilience) of neural networks to adversarial perturbations is an active topic of investigation. Without claiming to be exhaustive, this paper provides a brief overview of existing research directions aimed at improving safety and robustness of neural networks. Local (also called pointwise) robustness is defined with respect to an input point and its neighbourhood as the invariance of the classification over the neighbourhood. Global robustness is usually estimated as the expectation of local robustness over the test dataset weighted by the input distribution.

2.1 Heuristic search for adversarial examples

A number of approaches have been proposed to search for adversarial examples to exhibit their lack of robustness, typically by transforming the search into an optimisation problem, albeit without providing guarantees that adversarial examples do not exist if not found. In [17], search for adversarial examples is performed by minimising the L_2 distance between the images while maintaining the misclassification. Its improvement, Fast Gradient Sign Method (FGSM), uses a cost function to direct the search along the gradient. In [5], the optimisation problem proposed in [17] is adapted to attacks based on other norms, such as L_0 and L_∞. Instead of optimisation, JSMA [13] uses a loss function to create a "saliency map" of the image, which indicates the importance of each pixel in the classification decision. [19] introduces a game-based approach for finding adversarial examples by extracting the features of the input image using the SIFT [9] method. Then, working on a mixture of Gaussians representation of the image, the two players respectively select a feature and a pixel in the feature to search for an adversarial attack. This method is able to find the adversarial example in Figure 1 in a matter of seconds.

2.2 Automated verification approaches

In contrast to heuristic search for adversarial examples, verification approaches aim to provide *formal guarantees* on the robustness of DNNs. An early verification approach [14] encodes the entire network as a set of constraints and reduces the verification to the satisfiability problem. [8] improves on [14] by by extending the approach to work with piecewise linear ReLU functions, scaling up to networks with 300 ReLU nodes. [7] develops a verification framework that employs discretisation and a layer-by-layer refinement to exhaustively explore a finite region of the vector spaces associated with the input layer or the hidden layers, and scales to work with larger networks. [15] presents a verification approach based on computing the reachable set of outputs using global optimisation. In [12], techniques based on abstract interpretation are formulated, whereas [11] employ robust optimisation.

Several approaches analyse the robustness of neural networks by considering the maximal size of the perturbation that will not cause a misclassification. For a given input point, the *maximal safe radius* is defined as the largest radius centred on that point within which no adversarial examples exist. Solution methods include encoding as a set of constraints and reduction to satisfiability or optimisation [18]. In [20], the game-based approach of [19] is extended to anytime computation of upper and lower bounds on the maximum safe radius problem, providing a theoretical guarantee that it can reach the exact value. The method works by "gridding" the input space based on the Lipschitz constant and checking only the "corners" of the grid. Lower bound computation employs $A^\star$ search.

Since verification for state-of-the-art neural networks is an NP problem, testing methods that ensure high levels of coverage have also been developed [16].

2.3 Towards probabilistic verification for deep neural networks

All works listed above assume a trained network with fixed weights and therefore yield deterministic robustness guarantees. Since neural networks have a natural probabilistic interpretation, they lend themselves to frameworks for computing *probabilistic guarantees* on their robustness. Bayesian neural networks (BNNs) are neural networks with distributions over their weights, which can capture the uncertainty within the learning model [10]. The neural network can thus return an uncertainty estimate (typically computed pointwise, see [6]) along with the output, which is important for safety-critical applications.

In [3], *probabilistic robustness* is considered for BNNs, using a probabilistic generalisation of the usual statement of (deterministic) robustness to adversarial examples [7], namely the computation of the probability (induced by the distribution over the BNN weights) of the classification being invariant over the neighbourhood around a given input point. Since the computation of the posterior probability for a BNN is intractable, the method employs statistical model checking [21], based on the observation that each sample taken from the (possibly approximate) posterior weight distribution of the BNN induces a deterministic neural network. The latter can thus be analysed using existing verification techniques for deterministic networks mentioned above (e.g. [7, 8, 15]).

A related safety and robustness verification approach, which offers formal guarantees, has also been developed for Gaussian process (GP) models, for regression [4] and classification [2]. In contrast to DNNs, where trade offs between robustness and accuracy have been observed [11, 3], robustness of GPs increases with training. More research is needed to explore these phenomena.

3 Conclusion

The pace of development in Artificial Intelligence has increased sharply, stimulated by the advances and wide acceptance of the machine learning technology. Unfortunately, recent forays of technology companies into real-world applications have exposed the brittleness of deep learning. There is a danger that tacit acceptance of deep learning will lead to flawed AIs deployed in critical situations, at a considerable cost. Machine learning plays a fundamental role in enabling artificial agents, but developments so far have focused on 'narrow' AI tasks, such as computer vision and speech recognition, which lack the ability to reason about interventions, counterfactuals and 'what if' scenarios. To achieve "strong" AI, greater emphasis is necessary on rigorous modelling and verification technologies that support such reasoning, as well as development of novel synthesis techniques that guarantee the correctness of machine learning components by construction. Importantly, automated methods that provide probabilistic guarantees which properly take account of the learning process have a role to play and need to be investigated.

References

1 Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

2 Arno Blaas, Luca Laurenti, Andrea Patane, Luca Cardelli, Marta Kwiatkowska, and Stephen J. Roberts. Robustness Quantification for Classification with Gaussian Processes. *CoRR abs/1905.11876*, 2019. `arXiv:1905.11876`.

3 Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, Nicola Paoletti, Andrea Patane, and Matthew Wicker. Statistical Guarantees for the Robustness of Bayesian Neural Networks. In *IJCAI 2019*, 2018. See `arXiv:1809.06452`.

4 Luca Cardelli, Marta Kwiatkowska, Luca Laurenti, and Andrea Patane. Robustness guarantees for Bayesian inference with Gaussian processes. In *AAAI 2019*, 2018. See `arXiv:1809.06452`.

5 Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 39–57. IEEE, 2017.

6 Yarin Gal. *Uncertainty in deep learning.* PhD thesis, University of Cambridge, 2016.

7 Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *CAV*, pages 3–29. Springer, 2017.

8 Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient SMT solver for verifying deep neural networks. In *CAV*, pages 97–117. Springer, 2017.

9 David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

10 David JC MacKay. A practical Bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

11 A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards Deep Learning Models Resistant to Adversarial Attacks. *arXiv e-prints*, June 2017. `arXiv:1706.06083`.

12 Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable Abstract Interpretation for Provably Robust Neural Networks. In *ICML 2018*, pages 3578–3586, 2018.

13 Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*, pages 372–387. IEEE, 2016.

14 Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *CAV*, pages 243–257. Springer, 2010.

15 Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. In *IJCAI*, pages 2651–2659. AAAI Press, 2018.

16 Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic Testing for Deep Neural Networks. In *ASE 2018*, pages 109–119, 2018.

17 Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

18 Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating Robustness of Neural Networks with Mixed Integer Programming. *CoRR abs/1711.07356*, 2017. `arXiv:1711.07356`.

19 Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *TACAS*, pages 408–426. Springer, 2018.

20 Min Wu, Matthew Wicker, Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. A Game-Based Approximate Verification of Deep Neural Networks with Provable Guarantees. *Theoretical Computer Science*, 2018. To appear. See `arXiv:1807.03571`.

21 Håkan LS Younes, Marta Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer*, 8(3):216–228, 2006.

Synthesis of Safe, Optimal and Compact Strategies for Stochastic Hybrid Games

Kim G. Larsen
Department of Computer Science, Aalborg University, DK
kgl@cs.aau.dk

Abstract

Uppaal-Stratego is a recent branch of the verification tool Uppaal allowing for synthesis of safe and optimal strategies for stochastic timed (hybrid) games. We describe newly developed learning methods, allowing for synthesis of significantly better strategies and with much improved convergence behaviour. Also, we describe novel use of decision trees for learning orders-of-magnitude more compact strategy representation. In both cases, the seek for optimality does not compromise safety.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models; Theory of computation → Verification by model checking; Theory of computation → Online learning theory; Theory of computation → Reinforcement learning

Keywords and phrases Timed automata, Stochastic hybrid grame, Symbolic synthesis, Reinforcement learning, Q-learning, M-learning

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.2

Category Invited Paper

Funding Kim G. Larsen is partly funded by the ERC-Project LASSO.

1 UPPAAL Stratego

Cyber-physical systems are often safety-critical and hence strong guarantees on their safety are paramount. Besides, resource efficiency and the quality of the delivered service are strong requirements and the behavior needs also to be optimized with respect to these objectives, of course, within the bounds of what is still safe. In order to achieve this, controllers of such systems can be either implemented manually or automatically synthesized. In the former case, due to the complexity of the system, coming up with a controller that is safe is difficult, even more so with the additional optimization requirement. In the latter case, the synthesis may succeed with significantly less effort, though the requirement on both safety and optimality is still a challenge for current synthesis methods. However, due to the size of the systems, the produced controllers may be very complex, hard to understand, implement, modify, or even just output. Indeed, even for moderately sized systems, we can easily end up with gigabytes-long descriptions of their controllers (in the algorithmic context called strategies).

In [5, 6], we introduced Uppaal-Stratego, a branch of Uppaal allowing for synthesis of safe and optimal strategies for stochastic (priced) timed games (STG). The process of using Uppaal-Stratego is depicted in Fig. 1. First, the STG $\mathcal{G}$ is abstracted into a 2-player (non-stochastic) timed game $\mathcal{TG}$, ignoring any stochasticity of the behaviour. Next, the Uppaal-Tiga [3] is used to synthesize a safe strategy σ_{safe} for $\mathcal{TG}$ and the safety specification φ. After that, the safe strategy is applied on $\mathcal{G}$ to obtain $\mathcal{G} \upharpoonright \sigma_{safe}$. It is now possible to perform reinforcement learning on $\mathcal{G} \upharpoonright \sigma_{safe}$ in order to iteratively learn a sub-strategy σ_{fast} that will optimize the expectated value of given quantitative cost, given as a run-based expressions (formally defining a random variable over runs).

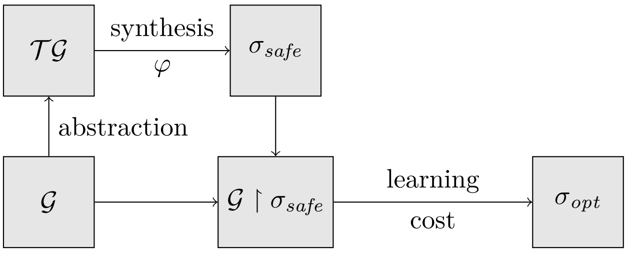


Figure 1 Uppaal-Stratego original workflow.

2 Better and Faster Learning

Though Uppaal-Stratego on a number of practical examples [10, 11, 13, 12, 7] has already demonstrated its ability to learn *near-optimal* strategies, we have recently improved Uppaal-Stratego in a number of ways. Firstly, the run-based reinforcement learning method used in Uppaal-Stratego is a continuous-time extension of the method in [8]. This method is known to be possibly caught in local optima and not necessarily converge towards the overall optimal strategy. In recent work [9], we show that we can significantly improve with respect to this existing method of Uppaal-Stratego both in terms of the quality of the the learned strategy, as well as in obtaining overall improved convergence characteristics as a function of the data size. The new learning methods in [9] are refinement based learning methods for continuous models: one based on Q-learning [15] and one related to Real Time Dynamic Programming [14, 2].

3 Compact Strategies

Aiming at using the synthesized strategies as control programs to be executed on small embedded platforms an important issue is how to encode compactly the synthesized strategies. For this purpose algorithmic methods have been devised to take into account both compositionality [4] as well as partial observability. Although neural network representations of strategies are attractive from a memory foot-print point of view, they may easily destroy the guarantee of safety. In [1], we introduce a new alternative method for learning compact representations of strategies in the form of *decision trees*. These decision trees are much smaller, more understandable, and can easily be exported as code that can be loaded into embedded systems. Despite the size compression and actual differences to the original strategy, we provide guarantees on both safety and optimality of the decision-tree strategy. On the top, we showed how to obtain yet smaller representations, which are still guaranteed safe, but achieve a desired trade off between size and optimality. Finally, we consider two case studies, one of them the cruise control from [13, 12], showing size reductions of two orders of magnitude, and quantify the additional size-performance trade-off.

We summarize the end-to-end work flow of Uppaal-Stratego+ for obtaining a safe, optimal and compact strategy from the model, a safety specification and an optimization query, see Fig. 2. Uppaal-Tiga is used to generate the most-permissive safety strategy σ_{safe} for the given safety specification φ. Now we can either use the standard Uppaal-Stratego workflow to generate the optimal strategy σ_{opt} and then learn a decision tree for this, as depicted on the right path of Fig. 2; or take the new approach following the left branch in Fig. 2. Here, we first learn a DT $\mathcal{T}^{k,p}_{\sigma_{safe}}$ from σ_{safe} using so-called minimum splitting size k and p rounds of safe pruning. This DT is smaller than the one representing σ_{safe} exactly,

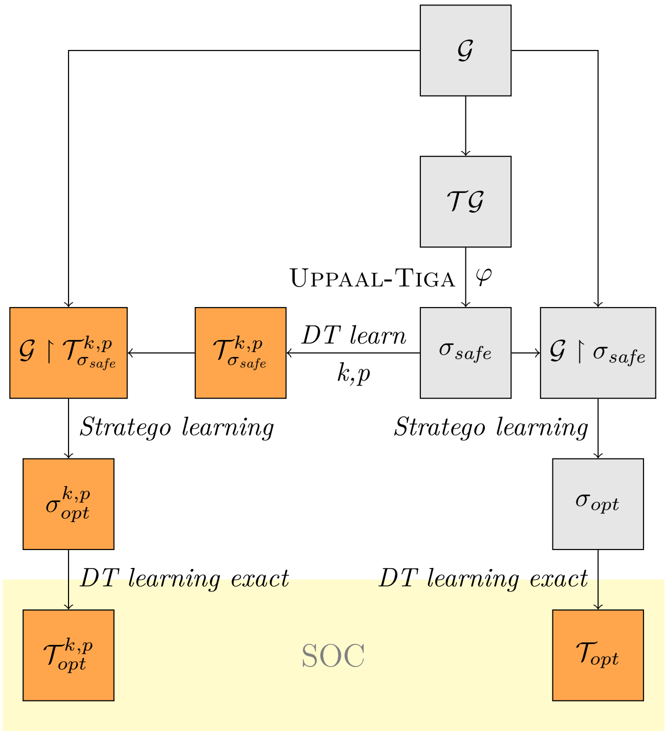


Figure 2 UPPAAL-STRATEGO+workflow. The dark orange nodes are the additions to the original workflow, which now involve DT learning, the yellow-shaded area delimits the desired safe, optimal, and compact strategy representations.

and the described strategy is less permissive. By restricting the game to this strategy and using UPPAAL-STRATEGO to get the optimal strategy, we get a smaller, but less performant strategy $\sigma^{k,p}_{opt}$ that is then output as DT $\mathcal{T}^{k,p}_{opt}$. In both cases, the resulting DT is safe by construction since we allow the DT to predict only pure actions (actions allowed by all configurations in a leaf). We convert these trees into a nested-if-statements-code, which can easily be loaded onto embedded systems.

4 On-line Learning

Finally, on-line methods for strategy synthesis/learning has been successfully applied in diverse domains such as heating systems [11] and intelligent traffic control [7]. The on-line method has the distinct advantages of not needing to store any strategy (as it is constantly computed during operation) but may be too slow to meet response-frequency of a given domain (e.g. in the order of milli-seconds for switched controllers for power electronics or adaptive cruice controls). Thus, we are investigating ways of making the on-line computations more efficient.

References

1 Pranav Ashok, Jan Kretinsky, Kim Guldstrand Larsen, Adrien Le Coent, Jakob Haahr Taankvist, and Maximilian Weininger. SOS: Safe, Optimal and Small Strategies for Stochastic Hybrid Games. In *To appear in Proceedings of QEST*, 2019.

2 Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to Act Using Real-time Dynamic Programming. *Artif. Intell.*, 72(1-2):81–138, January 1995. doi:10.1016/0004-3702(94)00011-O.

3 Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. UPPAAL-Tiga: Time for Playing Games! In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, volume 4590 of *Lecture Notes in Computer Science*, pages 121–125. Springer, 2007. doi:10.1007/978-3-540-73368-3_14.

4 Adrien Le Coënt, Laurent Fribourg, Nicolas Markey, Florian De Vuyst, and Ludovic Chamoin. Compositional synthesis of state-dependent switching control. *Theor. Comput. Sci.*, 750:53–68, 2018. doi:10.1016/j.tcs.2018.01.021.

5 Alexandre David, Peter Gjøl Jensen, Kim Guldstrand Larsen, Axel Legay, Didier Lime, Mathias Grund Sørensen, and Jakob Haahr Taankvist. On Time with Minimal Expected Cost! In Franck Cassez and Jean-François Raskin, editors, *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 129–145. Springer, 2014. doi:10.1007/978-3-319-11936-6_10.

6 Alexandre David, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marius Mikucionis, and Jakob Haahr Taankvist. Uppaal Stratego. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 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 9035 of *Lecture Notes in Computer Science*, pages 206–211. Springer, 2015. doi:10.1007/978-3-662-46681-0_16.

7 Andreas Berre Eriksen, Harry Lahrmann, and Kim Guldstrand Larsen andJakob Haahr Taankvist. Controlling Signalized Intersections using Machine Learning. In *World Conference on Transport Research*, 2019.

8 David Henriques, João Martins, Paolo Zuliani, André Platzer, and Edmund M. Clarke. Statistical Model Checking for Markov Decision Processes. In *Ninth International Conference on Quantitative Evaluation of Systems, QEST 2012, London, United Kingdom, September 17-20, 2012*, pages 84–93. IEEE Computer Society, 2012. doi:10.1109/QEST.2012.19.

9 Manfred Jaeger, Peter G. Jensen, Kim G. Larsen, Axel Legay, Sean Sedwards, and Jakob H. Taankvist. Teaching Stratego to Play Ball> Optimal Synthesis fo Continuous Space MDPs. In *To appear in Proceedings of ATVA*, 2019.

10 Shyam Lal Karra, Kim Guldstrand Larsen, Florian Lorber, and Jirí Srba. Safe and Time-Optimal Control for Railway Games. In Simon Collart Dutilleul, Thierry Lecomte, and Alexander B. Romanovsky, editors, *Reliability, Safety, and Security of Railway Systems. Modelling, Analysis, Verification, and Certification - Third International Conference, RSSRail 2019, Lille, France, June 4-6, 2019, Proceedings*, volume 11495 of *Lecture Notes in Computer Science*, pages 106–122. Springer, 2019. doi:10.1007/978-3-030-18744-6_7.

11 Kim G. Larsen, Marius Mikucionis, Marco Muñiz, Jirí Srba, and Jakob Haahr Taankvist. Online and Compositional Learning of Controllers with Application to Floor Heating. 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 244–259. Springer, 2016. doi:10.1007/978-3-662-49674-9_14.

12 Kim Guldstrand Larsen, Adrien Le Coënt, Marius Mikucionis, and Jakob Haahr Taankvist. Guaranteed Control Synthesis for Continuous Systems in Uppaal Tiga. In Roger D. Chamberlain, Walid Taha, and Martin Törngren, editors, *Cyber Physical Systems. Model-Based Design - 8th International Workshop, CyPhy 2018, and 14th International Workshop, WESE 2018, Turin, Italy, October 4-5, 2018, Revised Selected Papers*, volume 11615 of *Lecture Notes in Computer Science*, pages 113–133. Springer, 2018. doi:10.1007/978-3-030-23703-5_6.

13 Kim Guldstrand Larsen, Marius Mikucionis, and Jakob Haahr Taankvist. Safe and Optimal Adaptive Cruise Control. In Roland Meyer, André Platzer, and Heike Wehrheim, editors, *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015. Proceedings*, volume 9360 of *Lecture Notes in Computer Science*, pages 260–277. Springer, 2015. doi:10.1007/978-3-319-23506-6_17.

14 Alexander L. Strehl, Lihong Li, and Michael L. Littman. Incremental Model-based Learners With Formal Learning-Time Guarantees. *CoRR*, 2012. arXiv:1206.6870.

15 Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, 1989.

Program Invariants

Joël Ouaknine
Max Planck Institute for Software Systems, Saarbrücken, Germany
Oxford University, UK
https://people.mpi-sws.org/~joel/
joel@mpi-sws.org

Abstract

Automated invariant generation is a fundamental challenge in program analysis and verification, going back many decades, and remains a topic of active research. In this talk I'll present a select overview and survey of work on this problem, and discuss unexpected connections to other fields including algebraic geometry, group theory, and quantum computing. (No previous knowledge of these topics will be assumed.)

This is joint work with Ehud Hrushovski, Amaury Pouly, and James Worrell.

2012 ACM Subject Classification Theory of computation → Invariants

Keywords and phrases Automated invariant generation, program analysis and verification

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.3

Category Invited Talk

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 3; pp. 3:1–3:1
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Concurrent Algorithms and Data Structures for Model Checking

Jaco van de Pol
Aarhus University, Denmark
University of Twente, The Netherlands
`http://www.cs.au.dk/~jaco`
jaco@cs.au.dk

Abstract

Model checking is a successful method for checking properties on the state space of concurrent, reactive systems. Since it is based on exhaustive search, scaling this method to industrial systems has been a challenge since its conception. Research has focused on *clever data structures and algorithms*, to reduce the size of the state space or its representation; *smart search heuristics*, to reveal potential bugs and counterexamples early; and *high-performance computing*, to deploy the brute force processing power of clusters of compute-servers. The main challenge is to combine these approaches – brute-force alone (when implemented carefully) can bring a linear speedup in the number of processors. This is great, since it reduces model-checking times from days to minutes. On the other hand, proper algorithms and data structures can lead to exponential gains. Therefore, the *parallelization bonus* is only real if we manage to speedup clever algorithms.

There are some obstacles though: many linear-time graph algorithms depend on a depth-first exploration order, which is hard to parallelize. Examples include the detection of strongly connected components (SCC) and the nested depth-first-search (NDFS) algorithm. Both are used in model checking LTL properties. Symbolic representations, like binary decision diagrams (BDDs), reduce model checking to "pointer-chasing", leading to irregular memory-access patterns. This poses severe challenges on achieving actual speedup in (clusters of) modern multi-core computer architectures.

This talk presents some of the solutions found over the last 10 years, which led to the high-performance model checker LTSmin [2]. These include parallel NDFS (based on the PhD thesis of Alfons Laarman [3]), the parallel detection of SCCs with concurrent Union-Find (based on the PhD thesis of Vincent Bloemen [1]), and concurrent BDDs (based on the PhD thesis of Tom van Dijk [4]).

Finally, I will sketch a perspective on moving forward from high-performance model checking to *high-performance synthesis algorithms*. Examples include parameter synthesis for stochastic and timed systems, and strategy synthesis for (stochastic and timed) games.

2012 ACM Subject Classification Software and its engineering → Model checking; Theory of computation → Verification by model checking; Theory of computation → Parallel algorithms

Keywords and phrases model checking, parallel algorithms, concurrent datastructures

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.4

Category Invited Talk

References

1 Vincent Bloemen. *Strong Connectivity and Shortest Paths for Checking Models*. PhD thesis, University of Twente, Enschede, Netherlands, 2019. URL: `https://research.utwente.nl/en/publications/strong-connectivity-and-shortest-paths-for-checking-models`.

2 Gijs Kant, Alfons Laarman, Jeroen Meijer, Jaco van de Pol, Stefan Blom, and Tom van Dijk. LTSmin: High-performance language-independent model checking. In *TACAS*, volume 9035 of *Lecture Notes in Computer Science*, pages 692–707. Springer, 2015.

3 Alfons Laarman. *Scalable multi-core model checking*. PhD thesis, University of Twente, Enschede, Netherlands, 2014. URL: `http://eprints.eemcs.utwente.nl/25673/`.

4 Tom van Dijk. *Sylvan: multi-core decision diagrams*. PhD thesis, University of Twente, Enschede, Netherlands, 2016. URL: `http://purl.utwente.nl/publications/100676`.

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 4; pp. 4:1–4:1
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Of Cores: A Partial-Exploration Framework for Markov Decision Processes

Jan Křetínský
Technical University of Munich, Germany
jan.kretinsky@in.tum.de

Tobias Meggendorfer
Technical University of Munich, Germany
tobias.meggendorfer@in.tum.de

Abstract

We introduce a framework for approximate analysis of Markov decision processes (MDP) with bounded-, unbounded-, and infinite-horizon properties. The main idea is to identify a "core" of an MDP, i.e., a subsystem where we provably remain with high probability, and to avoid computation on the less relevant rest of the state space. Although we identify the core using simulations and statistical techniques, it allows for rigorous error bounds in the analysis. Consequently, we obtain efficient analysis algorithms based on partial exploration for various settings, including the challenging case of strongly connected systems.

2012 ACM Subject Classification Theory of computation → Verification by model checking; Theory of computation → Random walks and Markov chains

Keywords and phrases Markov Decision Processes, Reachability, Approximation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.5

Related Version A full version of the paper is available at `https://arxiv.org/abs/1906.06931`.

Funding This work has been partially supported by the Czech Science Foundation grant No. 18-11193S and the German Research Foundation (DFG) project KR 4890/2 "Statistical Unbounded Verification".

1 Introduction

Markov decision processes (MDP) are a well established formalism for modelling, analysis and optimization of probabilistic systems with non-determinism, with a large range of application domains [18]. Classical objectives such as reachability of a given state or the long-run average reward (mean payoff) can be solved by a variety of approaches. In theory, the most suitable approach is linear programming as it provides exact answers (rational numbers with no representation imprecision) in polynomial time. However, in practice for systems with more than a few thousand states, linear programming is not very usable, see, e.g., [2]. As an alternative, one can apply dynamic programming, typically value iteration (VI) [4], the default method in the probabilistic model checkers PRISM [13] and Storm [9].

Despite better practical scalability of VI, systems with more than a few million states still remain out of reach of the analysis not only because of time-outs, but now also memory-outs, see, e.g., [6]. There are various heuristics designed to deal with so large state spaces, including abstractions, e.g., [8, 11], or a dual approach based on restricting the analysis to a part of the state space. Examples of the latter approach are asynchronous VI in probabilistic planning, e.g., [17], or projections in approximate dynamic programming, e.g., [5]. In both, only a certain subset of states is considered for analysis, leading to speed ups in orders of magnitude. These are best-effort solutions, which can only guarantee convergence to the true result in the limit, with no error bounds at any finite time. Surprisingly, this was the case with the

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 5; pp. 5:1–5:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

standard VI as well until recently, when an error bound (and thus a stopping criterion) was given independently in [10, 6]. The error bound follows from the under- and (newly obtained) over-approximations converging to the true value. This resulted not only in error bounds on VI, but opened the door to error bounds for other techniques, including those where even convergence is not guaranteed. For instance, while VI iteratively approximates the value of all states, the above-mentioned *asynchronous VI* evaluates states at different paces. Thus convergence is often unclear and even the rate of convergence is unknown and very hard to analyze. However, it is not too hard to extend the error bound technique for VI to asynchronous VI. A prime example is the modification of BRTDP [17] to reachability [6] with error bounds. These ideas are further developed for, e.g., settings with long-run average reward [2] or continuous time [1].

While these solutions are efficient, they are ad-hoc, sharing the idea of *simulation / learning-based partial exploration* of the system, but not the correctness proof. In this paper, we build the foundations for designing such frameworks and provide a new perspective on these approaches, leading to algorithms for settings where previous ideas cannot apply.

The previous algorithms use (i) simulations to explore the state space and (ii) planning heuristics and machine learning to analyze the experience and to bias further simulations to areas that seem more relevant for the analysis of the given property (e.g., reaching a state s_{42}), where (iii) the exact VI computation takes place and yields results with a guaranteed error bound. In contrast, this paper identifies a general concept of a "*core*" of the MDP, independently of the particular objective (which states to reach) and, to a certain extent, even of the type of property (reachability, mean payoff, linear temporal logic formulae, etc.). This core intuitively consists of states that are important for the analysis of the MDP, whereas the remaining parts of the state space affect the result only negligibly. To this end, the defining property of a core is that the system stays within the core with high probability.

There are several advantages of cores, compared to the tailored techniques. Since the core is agnostic of any particular property, it can be *re-used* for multiple queries. Thus, the repetitive effort spent by the simulations and heuristics to explore the relevant parts of the state space by the previous algorithms can be saved. Furthermore, identifying the core can serve to better *understand* the typical behaviour of the system. Indeed, the core is typically a lot smaller than the system (and thus more amenable to understand) and only contains the more likely behaviours, even for real-world models as shown in the experimental evaluation. Finally, the general concept of core provides a *unified* argument for the correctness of the previous algorithms since – implicitly – they gradually construct a core. This abstract view thus allows for easier development of further partial-exploration techniques within this framework.

More importantly, making the notion of core explicit naturally leads us to identify a new standpoint and approach for the more complicated case of strongly connected systems, where the previous algorithms as well as cores cannot help. In technical terms, minimal cores are closed under end components. Consequently, the minimal core for a strongly connected system is the whole system. And indeed, it is impossible to give guarantees on infinite-horizon behaviour whenever a single state is ignored. In order to provide *some* feasible analysis for this case, we introduce the n-step core. It is defined by the system staying there with high probability for *some* time. However, instead of n-step analysis, we suggest to observe the "stability" of the core, i.e. the tendency of the probability to leave this core if longer and longer runs are considered. We shall argue that this yields (i) rigorous bounds for N-step analysis for $N \gg n$ more efficiently than a direct N-step analysis, and (ii) finer information on the "long run" behaviour (for different lengths) than the summary for the infinite run, which, n.b., never occurs in reality. This opens the door towards a rigorous analysis of "typical" behaviour of the system, with many possible applications in the design and interpretation of complex systems.

Our contribution can be summarized as follows:

- We introduce the notion of core, study its basic properties, in its light re-interpret previous results in a unified way, and discuss its advantages.
- We stipulate a new view on long-run properties as rather corresponding to long runs than an infinite one. Then a modified version of cores allows for an efficient analysis of strongly connected systems, where other partial-exploration techniques necessarily fail.
- We show how these modified cores can aid in design and interpretation of systems.
- We provide efficient algorithms for computing both types of cores and evaluate them on several examples.

2 Preliminaries

In this section, we recall basics of probabilistic systems and set up the notation. We assume familiarity with the central ideas of measure theory. As usual, $\mathbb{N}$ and $\mathbb{R}$ refers to the (positive) natural numbers and real numbers, respectively. For any set S, we use $\overline{S}$ to denote its complement. A *probability distribution* on a finite set X is a mapping $p : X \to [0, 1]$, such that $\sum_{x \in X} p(x) = 1$. Its *support* is denoted by $\mathrm{supp}(p) = \{x \in X \mid p(x) > 0\}$. $\mathcal{D}(X)$ denotes the set of all probability distributions on X. An event happens *almost surely* (a.s.) if it happens with probability 1.

▶ **Definition 1.** *A* Markov chain (MC) *is a tuple* $\mathsf{M} = (S, s_0, \delta)$*, where* S *is a countable set of* states*,* $s_0 \in S$ *is the* initial *state, and* $\delta : S \to \mathcal{D}(S)$ *is a* transition function *that for each state* s *yields a probability distribution over successor states.*

▶ **Definition 2.** *A* Markov decision process (MDP) *is a tuple of the form* $\mathcal{M} = (S, s_0, A, \mathsf{Av}, \Delta)$*, where* S *is a finite set of* states*,* $s_0 \in S$ *is the* initial *state,* A *is a finite set of* actions*,* $\mathsf{Av} : S \to 2^A \setminus \{\emptyset\}$ *assigns to every state a non-empty set of* available actions*, and* $\Delta : S \times A \to \mathcal{D}(S)$ *is a* transition function *that for each state* s *and (available) action* $a \in \mathsf{Av}(s)$ *yields a probability distribution over successor states. Furthermore, we assume w.l.o.g. that actions are unique for each state, i.e.* $\mathsf{Av}(s) \cap \mathsf{Av}(s') = \emptyset$ *for* $s \neq s'$ *(which can be achieved by replacing* A *with* $S \times A$ *and adapting* Av *and* Δ *appropriately).*

For ease of notation, we overload functions mapping to distributions $f : Y \to \mathcal{D}(X)$ by $f : Y \times X \to [0, 1]$, where $f(y, x) := f(y)(x)$. For example, instead of $\delta(s)(s')$ and $\Delta(s, a)(s')$ we write $\delta(s, s')$ and $\Delta(s, a, s')$, respectively.

An *infinite path* ρ in a Markov chain is an infinite sequence $\rho = s_0 s_1 \dots \in S^\omega$, such that for every $i \in \mathbb{N}$ we have that $\delta(s_i, s_{i+1}) > 0$. A *finite path* (or *history*) $\varrho = s_0 s_1 \dots s_n \in S^*$ is a finite prefix of an infinite path. Similarly, an *infinite path* in an MDP is some infinite sequence $\rho = s_0 a_0 s_1 a_1 \dots \in (S \times A)^\omega$, such that for every $i \in \mathbb{N}$, $a_i \in \mathsf{Av}(s_i)$ and $\Delta(s_i, a_i, s_{i+1}) > 0$. *Finite paths* ϱ are defined analogously as elements of $(S \times A)^* \times S$. We use ρ_i and ϱ_i to refer to the i-th state in the given (in)finite path.

A *strategy* on an MDP is a function $\pi : (S \times A)^* \times S \to \mathcal{D}(A)$, which given a finite path $\varrho = s_0 a_0 s_1 a_1 \dots s_n$ yields a probability distribution $\pi(\varrho) \in \mathcal{D}(\mathsf{Av}(s_n))$ on the actions to be taken next. We denote the set of all strategies of an MDP by Π. Fixing any strategy π induces a Markov chain $\mathcal{M}^\pi = (S^\pi, s_0^\pi, \delta^\pi)$, where the states are given by $S^\pi = (S \times A)^* \times S$ and, for some state $\varrho = s_0 a_0 \dots s_n \in S^\pi$, the successor distribution is defined as $\delta^\pi(\varrho, \varrho a_{n+1} s_{n+1}) = \pi(\varrho, a_{n+1}) \cdot \Delta(s_n, a_{n+1}, s_{n+1})$.

Any Markov chain M induces a unique measure $\mathbb{P}^{\mathsf{M}}$ over infinite paths [3, p. 758]. Assuming we fixed some MDP $\mathcal{M}$, we use $\mathbb{P}^{\pi}_{s}$ to refer to the probability measure induced by the Markov chain $\mathcal{M}^{\pi}$ with initial state s. Whenever π or s are clear from the context, we may omit them, in particular, $\mathbb{P}^{\pi}$ refers to $\mathbb{P}^{\pi}_{s_0}$. See [18, Sec. 2.1.6] for further details. For a given MDP $\mathcal{M}$ and measurable event E, we use the shorthand $\mathbb{P}^{\max}[E] := \sup_{\pi \in \Pi} \mathbb{P}^{\pi}[E]$ and $\mathbb{P}^{\max}_{s}[E] := \sup_{\pi \in \Pi} \mathbb{P}^{\pi}_{s}[E]$ to refer to the maximal probability of E over all strategies (starting in s). Analogously, $\mathbb{P}^{\min}[E]$ and $\mathbb{P}^{\min}_{s}[E]$ refer to the respective minimal probabilities.

A pair (T, B), where $\emptyset \neq T \subseteq S$ and $\emptyset \neq B \subseteq \bigcup_{s \in T} \mathsf{Av}(s)$, is an *end component* of an MDP $\mathcal{M}$ if (i) for all $s \in T, a \in B \cap \mathsf{Av}(s)$ we have $\operatorname{supp}(\Delta(s, a)) \subseteq T$, and (ii) for all $s, s' \in T$ there is a finite path $\varrho = s a_0 \dots a_n s' \in (T \times B)^* \times T$, i.e. the path stays inside T and only uses actions in B. Intuitively, an end component describes a set of states for which a particular strategy exists such that all possible paths remain inside these states. By abuse of notation, we identify an end component with the respective set of states, e.g., $s \in E = (T, B)$ means $s \in T$. An end component (T, B) is a *maximal end component (MEC)* if there is no other end component (T', B') such that $T \subseteq T'$ and $B \subseteq B'$. The set of MECs of an MDP $\mathcal{M}$ is denoted by $\mathsf{MEC}(\mathcal{M})$ and can be obtained in polynomial time [7].

In the following, we will primarily deal with unbounded and bounded variants of *reachability* queries. Essentially, for a given MDP and set of states, the task is to determine the maximal probability of reaching them, potentially within a certain number of steps. Technically, we are interested in determining $\mathbb{P}^{\max}[\Diamond T]$ and $\mathbb{P}^{\max}[\Diamond^{\leq n} T]$, where T is the set of target states and $\Diamond T$ ($\Diamond^{\leq n} T$) refers to the measurable set of runs that visit T at least once (in the first n steps). The dual operators $\Box T$ and $\Box^{\leq n} T$ refer to the set of runs which remain inside T forever or for the first n steps, respectively. See [3, Sec. 10.1.1] for further details. Our techniques are easily extendable to other related objectives like *long run average reward* (*mean payoff*) [18], *LTL formulae* or ω-regular objectives [3]. We briefly comment on this in Section 3.3.

We are interested in finding approximate solutions efficiently, i.e. trading precision for speed of computation. In our case, “approximate” means ε-optimal for some given precision $\varepsilon > 0$, i.e. the value we determine has an absolute error of less than ε. For example, given a reachability query $\mathbb{P}^{\max}[\Diamond T]$ and precision ε, we are interested in finding a value v with $|\mathbb{P}^{\max}[\Diamond T] - v| < \varepsilon$.

3 The Core Idea

In this section, we present the novel concept of *cores*, inspired by the approach of [6], where a specific reachability query was answered approximately through heuristic based methods. Omitted proofs can be found in [12, App. A.1]. We first establish a running example to motivate our work and explain the difference to previous approaches.

Consider a flight of an air plane. The controller, e.g. the pilot and the flight computer, can take many decisions to control the plane. The system as a whole can be in many different states. One may be interested in the maximal probability of arriving safely. This intuitively describes how likely it is to arrive, assuming that the pilot acts optimally and the computer is bug-free. Of course, this probability may be less than 100%, since some components may fail even under optimal conditions. See Figure 1 for a simplified MDP modelling this example.

The key observation in [6] is that some extreme situations may be very unlikely and we can simply assume the worst case for them without losing too much precision. This allows us to completely ignore these situations. For example, consider the unlikely event of hazardous bit flips during the flight due to cosmic radiation. This event might eventually lead to a

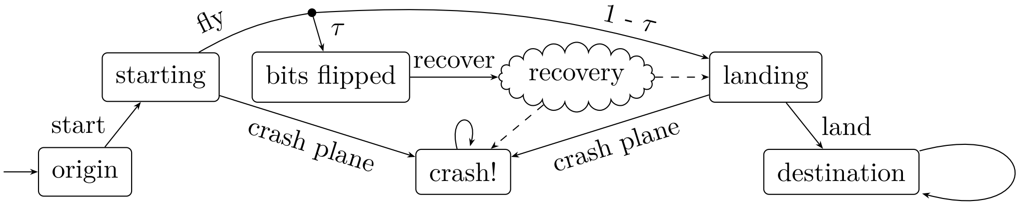


Figure 1 A simplified model of a flight, where $\tau = 10^{-10}$ is the probability of potentially hazardous bit flips occurring during the flight. The "recovery" node represents a complex recovery procedure, comprising many states.

crash or it might have no influence on the evolution of the system at all. Since this event is so unlikely to occur, we can simply assume that it always leads to a crash and still get a very precise result. Consequently, we do not need to explore the corresponding part of the state space (the "recovery" part), saving resources. As shown in the experimental evaluation in Section 5, many real-world models indeed exhibit a similar structure.

In [6], the state space was explored relative to a particular reachability objective, storing upper and lower bounds on each state for the objective in consideration. We make use of the same principle idea, but approach it from a different perspective, agnostic of any objective. We are interested in finding *all* relevant states of the system, i.e. all states which are reasonably likely to be reached. Such a set of states is an *intrinsic property* of the system, and we show that this set is both sufficient and necessary to answer *any* non-trivial reachability query ε-precisely. In particular, once computed, this set can be reused for multiple queries.

3.1 Infinite-Horizon Cores

First, we define the notion of an ε-*core*. Intuitively, an ε-core is a set of states which can only be exited with probability less than ε.

▶ **Definition 3** (Core). *Let $\mathcal{M}$ be an MDP and $\varepsilon > 0$. A set $S_\varepsilon \subseteq S$ is an ε-*core *if $\mathbb{P}^{\max}[\Diamond \overline{S_\varepsilon}] < \varepsilon$, i.e. the probability of ever exiting S_ε is smaller than ε.*

When ε is clear from the context, we may refer to an ε-core by "core". Observe that the core condition is equivalent to $\mathbb{P}^{\min}[\Box S_\varepsilon] \geq 1 - \varepsilon$.

The set of all states S trivially is a core for any ε. Naturally, we are interested in finding a core which is as small as possible, which we call a *minimal core*.

▶ **Definition 4** (Minimal Core). *Let $\mathcal{M}$ be an MDP and $\varepsilon > 0$. $S_\varepsilon^* \subseteq S$ is a* minimal ε-core *if it is inclusion minimal, i.e. S_ε^* is an ε-core and there exists no ε-core $S_\varepsilon' \subsetneq S_\varepsilon^*$.*

When ε is clear from the context, we may refer to a minimal ε-core by "minimal core". In the running example, a minimal core for $\varepsilon = 10^{-6}$ would contain all states except the "bit flipped" state and the "recovery" subsystem, since they are reached only with probability $\tau < \varepsilon$.

▶ Remark 5. We note that this idea may seem similar to the one of [19], but is subtly different. In that work, the authors consider a classical reachability problem using value iteration. They approximate the exit probability of a *fixed* set $S_?$ to bound the error on the computed reachability.

In the following, we derive basic properties of cores, show how to efficiently construct them, and relate them to the approaches of [2, 6].

First, we observe that finding a core of a given size (for a non-trivial ε) is NP-complete.

▶ **Theorem 6.** *For $0 < \varepsilon < \frac{1}{4}$, $\{(\mathcal{M}, k) \mid \mathcal{M}$ has an ε-core of size $k\}$ is NP-complete.*

Observe that this result only implies that finding minimal cores is hard. In the following section, we introduce a learning-based approach which quickly identifies reasonably sized cores.

▶ **Theorem 7.** *Let $\mathcal{M}$ be an MDP and $\varepsilon > 0$. A set $S_\varepsilon \subseteq S$ is an ε-core of $\mathcal{M}$ if and only if for every subset of states $R \subseteq S$ we have that $0 \leq \mathbb{P}^{\max}[\Diamond R] - \mathbb{P}^{\max}[\Diamond(R \cap S_\varepsilon) \cap \Box S_\varepsilon] < \varepsilon$.*

This theorem shows that for any reachability objective R, we can determine $\mathbb{P}^{\max}[\Diamond R]$ up to ε precision by determining the reachability of R on the sub-model induced by any ε-core, i.e. by only considering runs which remain inside S_ε. This also shows that, in general, cores are necessary to determine reachability up to precision ε.

We emphasize that this does *not* imply that identifying a core is necessary for all queries. For example, we have that $\mathbb{P}^{\max}_{s_0}[\Diamond\{s_0\}] = 1$ even without constructing any core. Nevertheless, for any non-trivial property, i.e. a reachability query with value less than $1-\varepsilon$, a computation restricted to a subset which does not satisfy the core property cannot give ε-guarantees on its results – only lower bounds can be proven. Thus, a set of states satisfying the core property has to be considered for non-trivial properties. In particular, the approach of [6] implicitly builds a core for such properties.

Of course, one could simply construct the whole state set S for the computation, which trivially satisfies the core condition. But, using the methods presented in the following section, we can efficiently identify a considerably smaller core. In particular, we observe in Section 5 that for some models we are able to identify a very small core orders of magnitude faster than the construction of the state set S, speeding up subsequent computations drastically.

3.2 Learning a Core

We introduce a sampling based algorithm which builds a core. In the interest of space, we only briefly describe the algorithm. Further discussion can be found in [12, App. A.2] and [6]. The algorithm is structurally very similar to the one presented in [6]. Nevertheless, we present it explicitly here since (i) it is significantly simpler and (ii) we introduce modifications later on.

We assume that the model is described by an initial state and a successor function, yielding all possible actions and the resulting distribution over successor states. This allows us to only construct a small fraction of the state space and achieve sub-linear runtime for some models.

During the execution of the algorithm, the system is traversed by following the successor function, starting from the initial state. Each state encountered is stored in a set of *explored* states, all other, not yet visited states are *unexplored.* Unexplored successors of explored states are called *partially explored.* Furthermore, the algorithm stores for each (explored) state s an upper bound $U(s)$ on the probability of reaching some unexplored state starting from s. The algorithm gradually grows the set of explored states and simultaneously updates these upper bounds, until the desired threshold is achieved in the initial state, i.e. $U(s_0) < \varepsilon$, and thus the set of explored states provably satisfies the core property. In particular, the upper bound is updated by sampling a path according to SamplePath and back-propagating the values along that path using Bellman backups.

SamplePath samples paths following some heuristic. In particular, it does not have to follow the transition probabilities given by the successor function. For example, a successor might be sampled with probability proportional to its upper bound times the transition

Algorithm 1 LearnCore.

```
Input: MDP $\mathcal{M}$, precision $\varepsilon > 0$, upper bounds $U$, state set $S_\varepsilon$ with $s_0 \in S_\varepsilon$
Output: $S_\varepsilon$ s.t. $S_\varepsilon$ is an $\varepsilon$-core
while $U(s_0) \geq \varepsilon$ do
    $\varrho \leftarrow$ SamplePath($s_0$, $U$)                              ▷ Generate path
    $S_\varepsilon \leftarrow S_\varepsilon \cup \varrho$                    ▷ Expand core
    UpdateECs($S_\varepsilon, U$)
    for $s$ in $\varrho$ in reverse order do                                 ▷ Back-propagate values
        $U(s) \leftarrow \max_{a \in A(s)} \sum_{s' \in S} \Delta(s, a, s') \cdot U(s')$
return $S_\varepsilon$
```

probability. The intuition behind this approach is that all states which are unlikely to be reached are not relevant and hence do not need to be included in the core. By trying to reach unexplored states the algorithm likely only reaches states which indeed are important.

UpdateECs identifies MECs of the currently explored sub-system and "collapses" them into a single representative state. This is necessary to ensure convergence of the upper bounds to the correct value – technically this process removes spurious fixed points of U.

For (a.s.) termination, we only require that the sampling heuristic is "(almost surely) fair". This means that (i) any partially explored state is reached eventually (a.s.), in order to explore a sufficient part of the state space, and (ii) any explored state with $U(s) > 0$ is visited infinitely often (a.s.), in order to back-propagate values accordingly. Further, we require that the initial upper bounds are consistent with the given state set, i.e. $U(s) \geq \mathbb{P}^{\max}_s[\Diamond \overline{S_\varepsilon}]$. This is trivially satisfied by $U(\cdot) = 1$. Note that in contrast to [6], the set whose reachability we approximate dynamically changes and, further, only upper bounds are computed.

▶ **Theorem 8.** *Algorithm 1 is correct and terminates (a.s.) if* SamplePath *is (a.s.) fair and the given upper bounds U are consistent with the given set S_ε.*

Proof Sketch. *Correctness*: By assumption $U(s)$ initially is a correct upper bound for the "escape" probability, i.e. $U(s) \geq \mathbb{P}^{\max}_s[\Diamond \overline{S_\varepsilon}]$. Each update (a Bellman backup) preserves correctness, independent of the sampled path. Hence, if $U(s_0) < \varepsilon$, we have $\mathbb{P}^{\max}_s[\Diamond \overline{S_\varepsilon}] < \varepsilon$.

Termination: As we assumed that SamplePath is (a.s.) fair, eventually (a.s.) the whole model will be explored, and all MECs will be collapsed by UpdateECs. Then, all states are visited infinitely often (a.s.), and thus all upper bounds will eventually converge to 0. ◀

As Algorithm 1 is correct and terminates for any faithful upper bounds and initial state set, we can restart the algorithm and interleave it with other approaches refining the upper bounds. For example, one could periodically update the upper bounds using, e.g., strategy iteration. Further, we can reuse the computed upper bounds and state set to compute a core for a tighter precision.

3.3 Using Cores for Verification

We explain how a core can be used for verification and how our approach differs from existing ones. Clearly, we can compute reachability or safety objectives on a given core ε-precisely. In this case, our approach is not too different from the one in [6]. Yet, we argue that our approach yields a stronger result. Due to cores being an intrinsic object, we are able to reuse and adapt this idea easily to many other objectives. Observe that a dedicated adaption may still yield slightly better performance, but requires significantly more work. For example, see [2] for an adaption to mean payoff.

To see how we can connect our idea to mean payoff, we briefly explain this objective and then recall an observation of [2]. First, rational rewards are assigned to each state, which are obtained on each visit. Then, the mean payoff of a particular run is the limit average reward obtained from the visited states. The mean payoff under a particular strategy then is obtained by integrating over the set of all runs. As mentioned by [2], a mean payoff objective can be decomposed into a separate analysis of each (explored) MEC and a (weighted) reachability query

$$\mathsf{optimal\ mean\ payoff} = \sup_{\pi \in \Pi} \sum\nolimits_{M \in \mathsf{MEC}(\mathcal{M})} \mathsf{mean\ payoff\ of\ } \pi \mathsf{\ in\ } M \cdot \mathbb{P}^{\pi}\left[\Diamond \Box M\right].$$

Since we can bound the reachability on unexplored MECs by the core property, we can easily bound the error on the computed mean payoff (assuming we know an a-priori lower and upper bound on the reward function). Consequently, we can approximate the optimal mean payoff by only analysing the corresponding core.

Similarly, LTL queries and parity objectives can be answered by a decomposition into analysis of MECs and their reachability. Intuitively, given a MEC one can decide whether the MEC is “winning” or “losing” for these objectives. The overall probability of satisfying the objective then equals the probability of reaching a winning MEC [3]. Again, we can bound the reachability of unexplored MECs and thus the error we incur when only analysing the core.

In general, many verification tasks can be decomposed into a reachability query and analysis of specific parts of the system. Since our framework is agnostic of the verification task in question, it can be transparently plugged in to obtain significant speed-ups.

We highlight that our approach is directly applicable to models with infinite state space, since finite cores still may exist for these models.

4 Beyond Infinite Horizon

In the previous section, we have seen that MECs play an essential role for many objectives. Hence, we study the interplay between cores and MECs.

▶ **Proposition 9.** *Let $M \in \mathsf{MEC}(\mathcal{M})$ be a MEC. If there is a state $s \in M$ with $\mathbb{P}^{\max}[\Diamond\{s\}] \geq \varepsilon$ then $M \subseteq S_\varepsilon$ for every ε-core S_ε.*

Proof. Recall that for $s, s' \in M$, we have $\mathbb{P}^{\max}_s[\Diamond\{s'\}] = 1$, thus $\mathbb{P}^{\max}[\Diamond\{s\}] = \mathbb{P}^{\max}[\Diamond\{s'\}] \geq \varepsilon$ and thus $s' \in S_\varepsilon$. ◀

This implies that sufficiently reachable MECs always need to be contained in a core *entirely*. Many models comprise only a few or even a single MEC, e.g., restarting protocols like mutual exclusion or biochemical models of reversible reactions. Together with the result of Theorem 7, i.e. constructing a core is necessary for ε-precise answers, this shows that in general we cannot hope for any reduction in state space, even when only requiring ε-optimal solutions for *any* of the discussed properties. In particular, the approach of [6] necessarily has to explore the full model. Yet, real-world models often exhibit a particular structure, with many states only being visited infrequently. Since we necessarily have to give up on something to obtain further savings, we propose an extension of our idea, motivated by a modification of our running example.

Instead of a one-way trip, consider the plane going back and forth between the origin and the destination, as shown in Figure 2. Clearly, the plane eventually will suffer from a bit flip, *independently* of the strategy. Furthermore, assuming that there is a non-zero probability of not being able to recover from the error, the plane will eventually crash.

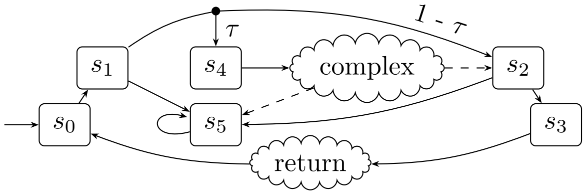


Figure 2 An adaptation of the model from Figure 1, with an added return trip, represented by the "return" node. State and action labels have been omitted in the interest of space.

We make two observations. First, any core needs to contain at least parts of the recovery sub-system, since it is reached with probability 1. Thus, this (complex) sub-system has to be constructed. Second, the witness strategy is meaningless, since any strategy is optimal – the crash cannot be avoided in the long run. In particular, deliberately crashing the plane has the same long run performance as flying it "optimally". In practice, we often actually are interested in the performance of such a model for a long, but not necessarily infinite horizon.

To this end, one could compute the step bounded variants of the objectives, but this incurs several problems: (i) choosing a sensible step bound n, (ii) computational overhead (a precise computation has a worst-case complexity of $|\Delta| \cdot n$ even for reachability), and (iii) all states reachable within n steps have to be constructed (which equals the whole state space for practically all models and reasonable choices of n). In the following, we present a different approach to this problem, again based on the idea of cores.

4.1 Finite-Horizon Cores

We introduce *finite-horizon cores*, which are completely analogous to (infinite-horizon) cores, only with a step bound attached to them.

▶ **Definition 10** (Finite-Horizon Core). *Let $\mathcal{M}$ be an MDP, $\varepsilon > 0$, and $n \in \mathbb{N}$. A set $S_{\varepsilon,n} \subseteq S$ is an* n-step ε-core *if $\mathbb{P}^{\max}[\Diamond^{\leq n}\overline{S_{\varepsilon,n}}] < \varepsilon$ and it is a* minimal n-step ε-core *if it is additionally inclusion minimal.*

As before, whenever n or ε are clear from the context, we may drop the corresponding part of the name. Again, similar properties hold and we omit the completely analogous proof.

▶ **Theorem 11.** *Let $\mathcal{M}$ be an MDP, $\varepsilon > 0$, and $n \in \mathbb{N}$. Then $S_{\varepsilon,n} \subseteq S$ is an n-step ε-core if and only if for all $R \subseteq S$ we have $0 \leq \mathbb{P}^{\max}[\Diamond^{\leq n} R] - \mathbb{P}^{\max}[\Diamond^{\leq n}(R \cap S_{\varepsilon,n}) \cap \Box^{\leq n} S_{\varepsilon,n}] < \varepsilon$.*

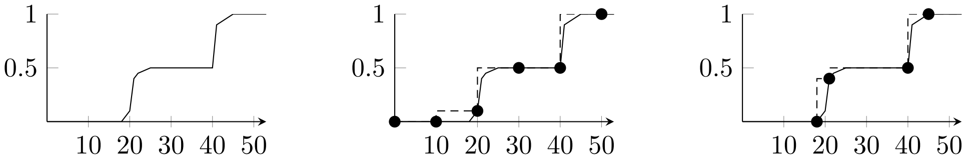
These finite-horizon cores are much smaller than their "infinite" counterparts on some models, even for large n. For instance, in our modified running example of Figure 2, omitting the "complex" states gives an n-step core even for very large n (depending on τ). On the other hand, finding such finite cores seems to be harder in practice. Naively, one could apply the core learning approach of Algorithm 1 to a modified model where the number of steps is encoded into the state space, i.e. $S' = S \times \{0, \dots, n\}$. Unfortunately, this yields abysmal performance, since we store and back-propagate $|S| \cdot n$ values instead of only $|S|$. Nevertheless, we can efficiently approximate them by enhancing our previous approach with further observations.

4.2 Learning a Finite Core

In Algorithm 2, we present our learning variant for the finite-horizon case. This algorithm is structurally very similar to the previous Algorithm 1. The fundamental difference is in Line 6, where the bounds are updated. One key observation is that the probability of

Algorithm 2 LearnFiniteCore.

```
Input: MDP $\mathcal{M}$, precision $\varepsilon > 0$, step bound $n$, upper bounds GetBound / UpdateBound,
    state set $S_{\varepsilon,n}$ with $s_0 \in S_{\varepsilon,n}$
Output: $S_{\varepsilon,n}$ s.t. $S_{\varepsilon,n}$ is an $n$-step $\varepsilon$-core
while GetBound$(s_0, n) \geq \varepsilon$ do
    $\varrho \leftarrow$ SamplePath$(s_0, n,$ GetBound$)$                          ▷ Generate path
    $S_{\varepsilon,n} \leftarrow S_{\varepsilon,n} \cup \varrho$                                  ▷ Update Core
    for $i \in [n-1, n-2, \dots, 0]$ do                                   ▷ Back-propagate values
        $s \leftarrow \varrho_i$, $r \leftarrow n - i$
        UpdateBound $\left(s, r, \max_{a \in A(s)} \sum_{s' \in S} \Delta(s,a,s') \cdot \text{GetBound}(s', r-1)\right)$
return $S_{\varepsilon,n}$
```



(a) True bounds.

(b) Simple approx.

(c) Adaptive approx.

Figure 3 An example for the different approximation approaches. The graphs depict the probability of exiting the core on the y axis within a given amount of steps on the x axis by a solid line and the corresponding approximation returned by GetBounds by a dashed line. From left to right, we have example bounds, which agree with the dense representation, followed by our sparse approach, which over-approximates the bounds, but requires less memory, and finally an adaptive approach, which closely resembles the precise bounds while consuming less memory.

reaching some set R within k steps is at least as high as reaching it within $k-1$ steps, i.e. $\mathbb{P}_s^{\max}[\Diamond^{\leq k} R] < \varepsilon$ is non-decreasing in k for any s and $R \subseteq S$. Therefore, we can use function over-approximations to store upper bounds sparsely and avoid storing n values for each state. To allow for multiple implementations, we thus delegate the storage of upper bounds to an abstract function approximation, namely GetBound and UpdateBound. This approximation scheme is supposed to store and retrieve the upper bound of reaching unexplored states for each state and number of *remaining* steps. We only require it to give a *consistent* upper bound, i.e. whenever we call UpdateBound(s, r, p), GetBound(s, r') will return at least p for all $r' \geq r$. Moreover, we require the trivial result GetBound$(s, 0) = 0$ for all states s. In the following Section 4.3, we list several possible instantiations.

▶ **Theorem 12.** *Algorithm 2 is correct if* UpdateBound*–*GetBound *are consistent and correct w.r.t. the given state set $S_{\varepsilon,n}$. Further, if* UpdateBound *stores all values precisely and* SamplePath *samples any state reachable within n steps infinitely often (a.s.), the algorithm terminates (a.s.).*

Sketch. *Correctness*: As before, the upper bound function is only updated through Bellman backups, which preserve correctness.

Termination: Given that the upper bound function stores all values precisely, the algorithm is an instance of asynchronous value iteration, which is guaranteed to converge [18]. ◀

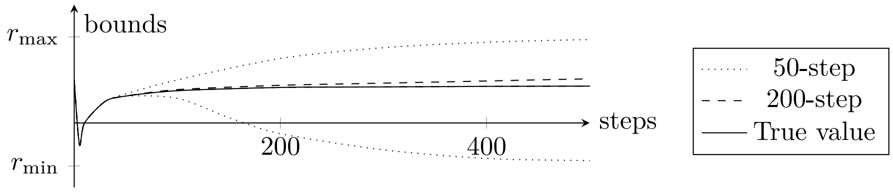


Figure 4 A schematic plot for an average reward extrapolation analysis on step bounded cores. The solid line represents the true value, while the dotted and dashed lines are the respective upper and lower bounds computed for a 50 and 200-step core. Note that the second dashed line (lower bound on the 200 core) coincides with the solid line (true value).

4.3 Implementing the function approximation

Several instances of the UPDATEBOUND–GETBOUND approach are outlined in Figure 3. The first, trivial implementation is dense storage, i.e. explicitly storing a table mapping $S \times \{0, \dots, n-1\} \to [0, 1]$. This table representation consumes an unnecessary amount of memory, since we do not need exact values in order to just guide the exploration. Hence, in our implementation, we use a simple sparse approach where we only store the value every K steps, where K manually chosen. This is depicted in Figure 3b for $K = 10$ – every black dot represents a stored value, the dashed lines represent the value returned by GETBOUNDS. The adaptive approach of Figure 3c adaptively chooses which values to store and is left for future work.

4.4 Stability and its applications

In this section, we explain the idea of a core's *stability*. Given an n-step core $S_{\varepsilon,n}$, we can easily compute the probability $\mathbb{P}^{\max}[\Diamond^{\leq N}\overline{S_{\varepsilon,n}}]$ of exiting the core within $N > n$ steps using, e.g., value iteration. The rate of increase of this exit probability intuitively gives us a measure of quality for a particular core. Should it rapidly approach 1 for increasing N, we know that the system's behaviour may change drastically within a few more steps. If instead this probability remains small even for large N, we can compute properties with a large step bound on this core with tight guarantees. We define stability as the whole function mapping the step bound N to the exit probability, since this gives a more holistic view on the system's behaviour than a singular value. In the following, we give an overview of how finite cores and the idea of stability can be used for analysis and interpretation, helping to design and understand complex systems.

As we have argued above, infinite-horizon properties may be deceiving, since (unrecoverable) errors often are bound to happen eventually. Consequently, one might be interested in a "very large"-horizon analysis instead. Unfortunately, such an analysis scales linearly both with the number of transitions and the horizon. Considering that many systems have millions of states, an analysis with a horizon of only 10,000 steps is already out of reach for existing tools. We first show how stable cores can be used for efficient extrapolation to such large horizons.

For simplicity, we consider reachability and argue how to transfer this idea to other objectives. We apply the ideas of interval iteration as used in, e.g., [10, 6], as follows. Intuitively, since we have no knowledge of the partially explored states, we simply assume the worst / best case for them, i.e. assign a lower bound of 0 and upper bound of 1. Furthermore, any explored target state is assigned a lower and upper bound of 1. By applying interval iteration, we can obtain bounds on the N-step and even unbounded reachability. Through the core property, the bounds for $N \leq n$ necessarily are smaller than ε. But, for larger N, there are no formal guarantees given by the core property. It might be the case that the core is left with probability 1 in $n+1$ steps. Nevertheless, in practice this allows us to get good approximations even for much larger bounds. Often the computation of an n-step core and subsequent approximation of the desired property is even faster than directly computing the N-step property, as shown in the evaluation.

For LTL and parity objectives, we can simply preprocess the obtained n-core by identifying the winning MECs and then applying the reachability idea, to obtain bounds on the satisfaction probability on the core. In the case of mean-payoff, we again require lower and upper bounds on the rewards $r_{\min}$ and $r_{\max}$ of the system in order to properly initialize the unknown values. Then, with the same approach, we can compute bounds on the n-step average reward by simply assign the lower and upper bounds $r_{\min}$ and $r_{\max}$ to all unexplored states instead of 0 and 1. See Figure 4 for a schematic plot of this analysis. Here, the 50-step core is too coarse for any reasonable analysis, it is unstable and can be exited with high probability. On the other hand, the 200-step core is very stable and accurately describes the system's behaviour for a longer period of time. Noticeably, it also contains a MEC guaranteeing a lower bound on the average reward, hence the lower bound actually agrees with the true value. Since the system may be significantly larger than the bounded cores or even infinitely large, this analysis potentially is much more efficient than analysis of the whole system, as shown in the experimental evaluation.

Note that we cannot use this method to obtain arbitrarily precise results. Given some n-step core and some (step bounded) property, there is a maximal precision we can achieve, depending on the property and the structure of the model. Hence, this method primarily is useful to quickly obtain an overview of a system's behaviour instead of verifying a particular property. As we have argued, one cannot avoid constructing a particular part of the state space in order to obtain an ε-precise result. Nevertheless, this may provide valuable insights in a system, quickly giving a good overview of its behaviour or potential design flaws.

We highlight that the presented algorithm can incrementally refine cores. For example, if a 100-step core does not yield a sufficiently precise extrapolation, the algorithm can reuse the computed core in order to construct a 200-step core. By applying this idea in an interactive loop, one can extract a condensed representation of the systems behaviour automatically, with the possibility for further refinements until the desired level of detail has been obtained.

5 Experimental Evaluation

In this section we give practical results for our algorithms on some examples, both the hand-crafted plane model and hand-picked models from case studies.

5.1 Implementation Details

We implemented our approach in Java, using PRISM [13] as a library for parsing its modelling language and basic computations. The implementation supports Markov chains, continuous-time Markov chains (CTMC, via embedding or uniformization [18, Ch. 11.5]) and Markov decision processes. Further, we implemented our own version of some utility classes, e.g., explicit MDP representation and MEC decomposition.

Inspired by the results of [6], we considered the following sampling heuristics. Given a state s, each heuristic first selects an action a which maximizes the expected upper bound. If there are multiple such actions, one of them is randomly selected. Then, a successor is chosen as follows: The RN (Random) heuristics samples a successors according

Table 1 Summary of our experimental results on several models and configurations for the infinite horizon core learning. The "PRISM" column shows the total number of states and construction time when explored with the explicit engine. The following columns show the size and total construction time of a 10^{-6}-core for each of the sampling heuristics.

Model	Param.	PRISM		RN		GD		MX	
`zeroconf`	$100; 5; 0.1$	496,291	8.4s	17,805	2.3s	1,072	0.4s	1,450	0.5s
(`N`; `K`; `loss`)	$100; 10; 0.1$	$3.0 \cdot 10^6$	55s	11,900	1.7s	1,006	0.3s	1,730	0.5s
	$100; 15; 0.1$	$4.7 \cdot 10^6$	159s	16,997	2.4s	1,067	0.4s	2,002	0.7s
`airplane`	100; `ff`	10,208	0.4s	6	0.1s	6	0.1s	6	0.1s
(`size`; `return`)	10000; `ff`	MEMOUT		6	0.1s	6	0.1s	6	0.1s
`brp`	20; 10	2,933	0.2s	2,352	0.3s	2,568	0.4s	2,675	0.5s
(`N`; `MAX`)	20; 100	26,423	0.6s	7,060	0.6s	3,421	0.4s	3,679	0.5s
	20; 1000	261,323	0.6s	7,624	0.6s	5,118	0.4s	3,912	0.5s
`wlan`	—	345,000	4.5s	344,835	52s	344,996	50s	344,997	52s

to $\Delta(s, a)$. The GD (Guided) approach samples a successor weighted by the respective upper bound, i.e. randomly select a state with probability proportional to $U(s') \cdot \Delta(s, a, s')$ or $\textsc{GetBound}(s', r) \cdot \Delta(s, a, s')$, respectively. Finally, MX (Max) samples a successor s' with probability proportional only to its upper bound $U(s')$ or $\textsc{GetBound}(s', r)$, respectively.

Recall that our algorithms can be restarted with faithful upper bounds and thus we can interleave it with other computations. In our implementation we alternate between the guided exploration of Algorithm 2 and precise computation on the currently explored set of states, guaranteeing convergence in the finite setting.

5.2 Models

In our evaluation, we considered the following models. All except the `airplane` model are taken from the PRISM case studies [16]. `airplane` is our running example from Figure 1 and Figure 2, respectively. The parameter `return` controls whether a return trip is possible, `size` quadratically influences the size of the "recovery" region. `zeroconf` [14] describes the IPv4 Zeroconf Protocol with `N` hosts, the number `K` of probes to send, and a probability of a message `loss`. `wlan` [15] is a model of two WLAN stations in a fixed network topology sending messages on the shared medium. `brp` is a DTMC modelling a file transfer of `N` chunks with bounded number `MAX` of retries per chunk. Finally, `cyclin` is a CTMC modelling the cell cycle control in eukaryotes with `N` molecules. We analyse this model using uniformization.

5.3 Results

We evaluated our implementation on an i7-4700MQ 4x2.40 GHz CPU with 16 GB RAM. We used a default precision of 10^{-6} for all experiments. The results for the infinite and finite construction are summarized in Table 1 and Table 2, respectively. We briefly discuss them in the following sections. Note that the results may vary due to the involved randomization.

5.3.1 Infinite Cores

As already explained in [6], the `zeroconf` model is very well suited for this type of analysis, since a lot of the state space is hardly reachable. In particular, most states are a result of collisions and several message losses, which is very unlikely. Consequently, a very small part

Table 2 Summary of our experimental results on several models and configurations for the finite-horizon 10^{-6}-core learning with 100 step bound; using the notation from Table 1.

Model	Param.	PRISM		RN		GD		MX	
`airplane`	100; `tt`	20,413	0.6s	11	0.3s	11	0.3s	554	0.6s
(`size`; `return`)	10000; `tt`	MEMOUT		11	0.3s	11	0.3s	603	0.6s
`wlan`	—	345,000	4.4s	36,718	15s	37,284	11s	36,825	12s
`cyclin`	4	431,101	12s	9,122	1,649s	4,380	3.9s	99,325	93s
(`N`)	5	$2.3 \cdot 10^6$	78s	26,925	8,840s	11,419	13s	817,058	1,462s

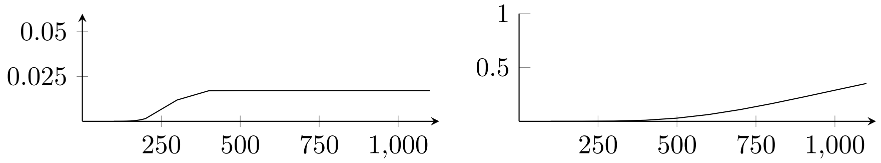


Figure 5 Stability analysis of the `wlan` (left) and `cyclin(N` $= 4$`)` (right) 100-step core built with the GD heuristic. The graphs show the probability of exiting the respective core within the given amount of steps. Note that the y axis of the `wlan` graph is scaled for readability.

of the model already satisfies the core property. In particular, the size of the core remains practically constant when increasing the parameter K, as only unimportant states are added to the system. Observe that the order of magnitude of explored states is very similar to the experiments from [6]. The same holds true for the `airplane` model, where a significant number of states is dedicated to recovering from an unlikely error. Hence, a small core exists independently of the total size of the model. The `brp` model shows applicability of the approach to Markov chains. In line with the other results, when scaling up the maximal number of allowed errors, the size of the core changes sub-linearly, since repeated errors are increasingly unlikely. In case of the `wlan` model, we observe that all our methods essentially construct the full model.

Comparison to [6]. We also executed the tool presented in [6] where applicable (only MDP are supported). We tested the tool both with a bogus `false` property, i.e. approximating the probability of reaching the empty set, which corresponds to constructing a core, and an actual property. We used the `MAX_DIFF` heuristic of [6], which is similar to GD. Especially on the `false` property, our tool consistently outperformed the previous one in terms of time and memory by up to several orders of magnitude. We suspect that this is mostly due to a more efficient implementation. The number of explored states was similar, as to be expected in light of Theorem 7 and its discussed consequences.

5.3.2 Finite Cores

As expected, the finite core construction yields good results on the `airplane` model, constructing only a small fraction of the state space. Interestingly, the MX heuristic explores significantly more states, which is due to this heuristic ignoring probabilities when selecting a successor and thus sampling a few paths in the recovery region. Also on the real-world models `wlan` and `cyclin`, the constructed 100-step core is significantly smaller than the whole model. For `wlan`, the construction of the respective cores unfortunately takes longer than building the whole model. We conjecture that a more fine-tuned implementation can

Method	Time		States
	model	bounds	
50 steps	0.4s	1.0s	2,137
100 steps	0.9s	3.5s	6,151
200 steps	4.0s	15.3s	22,989
Complete	8.7s	242.4s	431,101

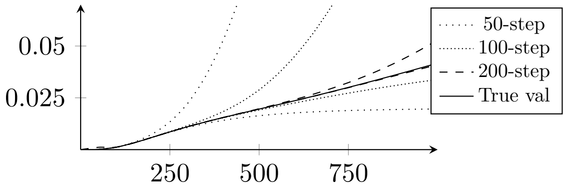


Figure 6 Overview of an extrapolation analysis for `cyclin`($N = 4$). We computed several step-bounded cores with precision 10^{-3}. On these, we computed bounds of a reachability query with increasing step bound. The table on the left lists the time for model construction + computation of the bounds for 1000 steps and the size of the constructed model. The plot on the right shows the upper and lower bounds computed for each core together with the true value. Observe that for growing step-size of the core, the approximation naturally gets more precise.

overcome this issue. In any case, model checking on the explored sub-system supposedly terminates significantly faster since only a much smaller state space is investigated, and the core can be re-used for more queries.

Finally, we applied the idea of stability from Section 4.2 on the `wlan` and `cyclin` models, with results outlined in Figure 5. Interestingly, for the `wlan` model, the escape probability stabilizes at roughly 0.017 and we obtain the exact same probability for *all* heuristics, even for $N = 10{,}000$. This suggests that by building the 100-step core we identified a very stable sub-system of the whole model. Additionally, we observe that at 200-400 steps, the behaviour of the system significantly changes. For the `cyclin` model, we instead observe a continuous rise of the exit probability. Nevertheless, even with 500 additional steps, the core still is only exited with a probability of roughly 6% and thus closely describes the system's behaviour.

On the `cyclin` model, we also applied our idea of extrapolation. The results are summarized in Figure 6. To show how performant this approach is, we reduced the precision of the core computation to 10^{-3}. Despite this coarse accuracy, we are able to compute accurate bounds on a 1000-step reachability query over 10 times faster by only building the 200-step core instead of constructing the full model. These results suggest that our idea of using the cores for extrapolation in order to quickly gain understanding of a model has a vast potential.

5.3.3 Heuristics

Overall, we see that the unguided, random sampling heuristic RN often is severely outperformed by the guided approaches GD and MX, both in terms of runtime and constructed states. Surprisingly, the differences between GD and MX often are small, considering that MX is significantly more "greedy" by completely ignoring the actual transition probabilities. We conjecture that this greediness is the reason for the abysmal performance of MX on the `cyclin` model, where GD seems to strike the right balance between exploration and exploitation. Altogether, the results show that a sophisticated heuristic increases performance by orders of magnitude and further research towards optimizing these heuristic may prove beneficial.

6 Conclusion

We have presented a new framework for approximate verification of probabilistic systems via partial exploration and applied it to both Markov chains and Markov decision processes. Our evaluation shows that, depending on the structure of the model, this approach can yield significant state space savings and thus reduction in model checking times. Our central idea – finding relevant sub-parts of the state space – can easily be extended to further models, e.g., stochastic games, and objectives, e.g., mean payoff. We have also shown how this idea can be transferred to the step-bounded setting and derived the notion of stability. This in turn allows for an efficient analysis of long-run properties and strongly connected systems.

Future work includes implementing a more sophisticated function approximation for the step-bounded case, e.g., as depicted in Figure 3c. Here, an adaptive method could yield further insight in the model by deriving points of interest, i.e. an interval of remaining steps where the exit probability significantly changes. These breakpoints might indicate a significant change in the systems behaviour, e.g., the probability of some error occurring not being negligible any more, yielding interesting insights into the structure of a particular model. For example, in the bounds of Figure 3, the regions around 20 and 40 steps, respectively, seems to be of significance.

Moreover, a more sophisticated sampling heuristic to be used in SamplePath could be of interest. For example, one could apply an advanced machine learning technique here, which also considers state labels or previous decisions and their outcomes.

In the spirit of [6], our approach also could be extended to a PAC algorithm for black-box systems. Extensions to stochastic games and continuous time systems are also possible.

Further interesting variations are cores for discounted objectives [20] or cost-bounded cores, a set of states which is left with probability smaller than ε given that at most k cost is incurred. This generalizes both the infinite (all edges have cost 0) and the step bounded cores (all edges have cost 1) and allows for a wider range of analysis.

References

1 Pranav Ashok, Yuliya Butkova, Holger Hermanns, and Jan Křetínskỳ. Continuous-Time Markov Decisions Based on Partial Exploration. In *ATVA*, pages 317–334. Springer, 2018.

2 Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Kretínský, and Tobias Meggendorfer. Value Iteration for Long-Run Average Reward in Markov Decision Processes. In *CAV*, pages 201–221, 2017. doi:10.1007/978-3-319-63387-9_10.

3 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

4 Richard Bellman. A Markovian decision process. *Journal of Mathematics and Mechanics*, pages 679–684, 1957.

5 Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control, Vol. II: Approximate Dynamic Programming*. Athena Scientific, 2012.

6 Tomás Brázdil, Krishnendu Chatterjee, Martin Chmelik, Vojtech Forejt, Jan Křetínský, Marta Z. Kwiatkowska, David Parker, and Mateusz Ujma. Verification of Markov Decision Processes Using Learning Algorithms. In *ATVA*, pages 98–114. Springer, 2014. doi:10.1007/978-3-319-11936-6_8.

7 Costas Courcoubetis and Mihalis Yannakakis. The Complexity of Probabilistic Verification. *J. ACM*, 42(4):857–907, 1995. doi:10.1145/210332.210339.

8 Pedro R. D'Argenio, Bertrand Jeannet, Henrik Ejersbo Jensen, and Kim Guldstrand Larsen. Reduction and Refinement Strategies for Probabilistic Analysis. In *PAPM-PROBMIV*, pages 57–76. Springer, 2002.

9 Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *CAV*, pages 592–600. Springer, 2017.

10 Serge Haddad and Benjamin Monmege. Reachability in MDPs: Refining convergence of value iteration. In *International Workshop on Reachability Problems*, pages 125–137. Springer, 2014.

11 Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. PASS: abstraction refinement for infinite probabilistic models. In *TACAS*, pages 353–357. Springer, 2010.

12 Jan Křetínský and Tobias Meggendorfer. Of Cores: A Partial-Exploration Framework for Markov Decision Processes. *arXiv e-prints*, June 2019. arXiv:1906.06931.

13 M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *TOOLS*, pages 200–204, 2002.

14 Marta Kwiatkowska, Gethin Norman, David Parker, and Jeremy Sproston. Performance analysis of probabilistic timed automata using digital clocks. *FMSD*, 29(1):33–78, 2006.

15 Marta Kwiatkowska, Gethin Norman, and Jeremy Sproston. Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In *Process Algebra and Probabilistic Methods: Performance Modeling and Verification*, pages 169–187. Springer, 2002.

16 Marta Z. Kwiatkowska, Gethin Norman, and David Parker. The PRISM Benchmark Suite. In *QEST*, pages 203–204. IEEE Computer Society, 2012. The models are accessible at http://www.prismmodelchecker.org/casestudies/.

17 H Brendan McMahan, Maxim Likhachev, and Geoffrey J Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *ICML*, pages 569–576. ACM, 2005.

18 M.L. Puterman. *Markov decision processes: Discrete stochastic dynamic programming.* John Wiley and Sons, 1994.

19 Tim Quatmann and Joost-Pieter Katoen. Sound Value Iteration. In *CAV (1)*, volume 10981 of *LNCS*, pages 643–661. Springer, 2018.

20 Aaron Sidford, Mengdi Wang, Xian Wu, and Yinyu Ye. Variance Reduced Value Iteration and Faster Algorithms for Solving Markov Decision Processes. In *SODA*, pages 770–787. SIAM, 2018.

Combinations of Qualitative Winning for Stochastic Parity Games

Krishnendu Chatterjee
IST Austria, Klosterneuburg, Austria

Nir Piterman
University of Gothenburg, Sweden
University of Leicester, UK

Abstract

We study Markov decision processes and turn-based stochastic games with parity conditions. There are three qualitative winning criteria, namely, sure winning, which requires all paths to satisfy the condition, almost-sure winning, which requires the condition to be satisfied with probability 1, and limit-sure winning, which requires the condition to be satisfied with probability arbitrarily close to 1. We study the combination of two of these criteria for parity conditions, e.g., there are two parity conditions one of which must be won surely, and the other almost-surely. The problem has been studied recently by Berthon et al. for MDPs with combination of sure and almost-sure winning, under infinite-memory strategies, and the problem has been established to be in NP$\cap$co-NP. Even in MDPs there is a difference between finite-memory and infinite-memory strategies. Our main results for combination of sure and almost-sure winning are as follows: (a) we show that for MDPs with finite-memory strategies the problem is in NP $\cap$ co-NP; (b) we show that for turn-based stochastic games the problem is co-NP-complete, both for finite-memory and infinite-memory strategies; and (c) we present algorithmic results for the finite-memory case, both for MDPs and turn-based stochastic games, by reduction to non-stochastic parity games. In addition we show that all the above complexity results also carry over to combination of sure and limit-sure winning, and results for all other combinations can be derived from existing results in the literature. Thus we present a complete picture for the study of combinations of two qualitative winning criteria for parity conditions in MDPs and turn-based stochastic games.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Two Player Games, Stochastic Games, Parity Winning Conditions

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.6

Related Version A full version of the paper is available [18], `https://arxiv.org/abs/1804.03453`.

Funding *Krishnendu Chatterjee*: This research is funded by Austrian Science Fund (FWF) NFN Grant S11407-N23 (RiSE/SHiNE), and Vienna Science and Technology Fund (WWTF) Project ICT15-003.
Nir Piterman: This research is funded by the ERC consolidator grant D-SynMA under the European Union's Horizon 2020 research and innovation programme (grant agreement No 772459).

1 Introduction

Stochastic games and parity conditions. Two-player games on graphs are an important model to reason about reactive systems, such as, reactive synthesis [21, 32] and open reactive systems [2]. To reason about probabilistic behaviors of reactive systems, such games are enriched with stochastic transitions, and this gives rise to models such as Markov decision processes (MDPs) [25, 33] and turn-based stochastic games [22]. While these games provide the model for stochastic reactive systems, the specifications for such systems that describe the

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 6; pp. 6:1–6:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Summary of Results for Sure-Almost-sure as well as Sure-Limit-sure Winning for Parity Conditions. New results are boldfaced. The reductions give algorithmic results from algorithms for non-stochastic games.

Model	Finite-memory	Infinite-memory
MDPs	**NP ∩ co-NP** **Reduction to non-stochastic parity games**	NP ∩ co-NP [5]
Turn-based stochastic game	**co-NP-complete** **Reduction to non-stochastic games** **with conjunction of parity conditions**	**co-NP-complete**

desired non-terminating behaviors are typically ω-regular conditions [35]. The class of parity winning conditions can express all ω-regular conditions, and has emerged as a convenient and canonical specification for algorithmic studies in the analysis of stochastic reactive systems.

Qualitative winning criteria. In the study of stochastic games with parity conditions, there are three basic qualitative winning criteria, namely, (a) *sure winning*, which requires all possible paths to satisfy the parity condition; (b) *almost-sure winning*, which requires the parity condition to be satisfied with probability 1; and (c) *limit-sure winning*, which requires the parity condition to be satisfied with probability arbitrarily close to 1. For MDPs and turn-based stochastic games with parity conditions, almost-sure winning coincides with limit-sure winning, however, almost-sure winning is different from sure winning [9]. Moreover, for all the winning criteria above, if a player can ensure winning, she can do so with memoryless strategies, that do not require to remember the past history of the game. All the above decision problems belong to NP ∩ co-NP, and the existence of polynomial-time algorithm is a major open problem.

Combination of multiple conditions. While traditionally MDPs and stochastic games have been studied with a single condition with respect to different winning criteria, in recent studies combinations of winning criteria has emerged as an interesting problem. An example is the *beyond worst-case synthesis* problem that combines the worst-case adversarial requirement with probabilistic guarantee [7]. Consider the scenario that there are two desired conditions, one of which is critical and cannot be compromised at any cost, and hence sure winning must be ensured, whereas for the other condition the probabilistic behavior can be considered. Since almost-sure and limit-sure provide the strongest probabilistic guarantee, this gives rise to stochastic games where one condition must be satisfied surely, and the other almost-surely (or limit-surely). The setting of two objectives have been considered in several prior works; such as in [1], where the primary objective is parity objective and the secondary objective is a quantitative mean-payoff objective; and in [5], where both the primary and the secondary objectives are different parity objectives, but for MDPs.

Previous results and open questions. While MDPs and turn-based stochastic games with parity conditions have been widely studied in the literature (e.g., [23, 24, 3, 14, 15, 9]), the study of combination of different qualitative winning criteria is recent. The problem has been studied only for MDPs with sure winning criteria for one parity condition, and almost-sure winning criteria (also probabilistic threshold guarantee) for another parity condition, and it has been established that even in MDPs infinite-memory strategies are required, and the decision problem lies in NP ∩ co-NP [5]. While the existence of infinite-memory strategies

Table 2 Conjunctions of various qualitative winning criteria.

Criterion 1	Criterion 2	Solution Method
Sure ψ_1	Sure ψ_2	Sure $(\psi_1 \wedge \psi_2)$
Sure ψ_1	Almost-sure ψ_2	This work
Sure ψ_1	Limit-sure ψ_2	This work
Almost-sure ψ_1	Almost-sure ψ_2	Almost-sure $(\psi_1 \wedge \psi_2)$
Almost-sure ψ_1	Limit-sure ψ_2	Almost-sure $(\psi_1 \wedge \psi_2)$
Limit-sure ψ_1	Limit-sure ψ_2	Almost-sure $(\psi_1 \wedge \psi_2)$

represent the general theoretical problem, many important questions have been left open for the problem where both objectives are parity objectives. For example, (i) the analysis for games, which is relevant in reactive synthesis, and (ii) finite-memory strategy synthesis, which represents the synthesis of practical controllers (such as Mealy or Moore machines). In this work we present answers to these open questions, with optimal complexity results.

Our results. In this work our main results are as follows:

1. For MDPs with finite-memory strategies, we show that the combination of sure winning and almost-sure winning for parity conditions also belong to NP $\cap$ co-NP, and we present a linear reduction to parity games. Our reduction implies a quasi-polynomial time algorithm, and also polynomial time algorithm as long as the number of indices for the sure winning parity condition is logarithmic. Note that no such algorithmic result is known for the infinite-memory case for MDPs.
2. For turn-based stochastic games, we show that the combination of sure and almost-sure winning for parity conditions is a co-NP-complete problem, both for finite-memory as well as infinite-memory strategies. For the finite-memory strategy case we present a reduction to non-stochastic games with conjunction of parity conditions, which implies a fixed-parameter tractable algorithm, as well as a polynomial-time algorithm as long as the number of indices of the parity conditions are logarithmic.
3. Finally, while for turn-based stochastic parity games almost-sure and limit-sure winning coincide, we show that in contrast, while ensuring one parity condition surely, limit-sure winning does not coincide with almost-sure winning even for MDPs. However, we show that all the above complexity results established for combination of sure and almost-sure winning also carry over to sure and limit-sure winning.

Our main results are summarized in Table 1. In addition to our main results, we also argue that our results complete the picture of all possible conjunctions of two qualitative winning criteria as follows: (a) conjunctions of sure (or almost-sure) winning with conditions ψ_1 and ψ_2 is equivalent to sure (resp., almost-sure) winning with the condition $\psi_1 \wedge \psi_2$ (the conjunction of the conditions); (b) by determinacy and since almost-sure and limit-sure winning coincide for ω-regular conditions, if the conjunction of $\psi_1 \wedge \psi_2$ cannot be ensured almost-surely, then the opponent can ensure that at least one of them is falsified with probability bounded away from zero; and thus conjunction of almost-sure winning with limit-sure winning, or conjunctions of limit-sure winning coincide with conjunction of almost-sure winning. This is illustrated in Table 2 and shows that we present a complete picture of conjunctions of two qualitative winning criteria in MDPs and turn-based stochastic games. Full proofs are available in a technical report [18].

Related work. We have already mentioned the most important related works above. We discuss other related works here. MDPs with multiple Boolean as well as quantitative objectives have been widely studied in the literature [17, 24, 26, 6, 16]. For non-stochastic games combination of various Boolean objectives is conjunction of the objectives, and such games with multiple quantitative objectives have been studied in the literature [36, 11]. For turn-based stochastic games, the general analysis of multiple quantitative objectives is intricate, and they have been only studied for special cases, such as, reachability objectives [20] and almost-sure winning [4, 10]. However none of these above works consider combinations of qualitative winning criteria. The problem of beyond worst-case synthesis has been studied for MDPs with various quantitative objectives [7, 34], such as long-run average, shortest path, and for parity objectives [5]. In particular [5] studies the problem of satisfying one parity objective surely and maximizing the probability of satisfaction of another parity objective in MDPs with infinite-memory strategies. We extend the literature of the study of beyond worst-case synthesis problem for parity objectives by considering combinations of qualitative winning in both MDPs and turn-based stochastic games, and the distinction between finite-memory and infinite-memory strategies. Thus in contrast to [5] we do not consider optimal probability of satisfaction, but consider turn-based stochastic games as well as finite-memory strategies.

2 Background

For a countable set S let $\mathcal{D}(S) = \{d : S \to [0,1] \mid \exists T \subseteq S$ such that $|T| \in \mathbb{N}, \forall s \notin T \,.\, d(s) = 0$ and $\Sigma_{s\in T} d(s) = 1\}$ be the set of discrete probability distributions with finite support over S. A distribution d is *pure* if there is some $s \in S$ such that $d(s) = 1$.

A stochastic turn-based game is $G = (V, (V_0, V_1, V_p), E, \kappa)$, where V is a finite set of configurations, V_0, V_1, and V_p form a partition of V to Player 0, Player 1, and stochastic configurations, respectively, $E \subseteq V \times V$ is the set of edges, and $\kappa : V_p \to \mathcal{D}(V)$ is a probabilistic transition for configurations in V_p such that $\kappa(v, v') > 0$ implies $(v, v') \in E$. If either $V_0 = \emptyset$ or $V_1 = \emptyset$ then G is a Markov Decision Process (MDP). If both $V_0 = \emptyset$ and $V_1 = \emptyset$ then G is a Markov Chain (MC). If $V_p = \emptyset$ then G is a turn-based game (non-stochastic). For an MC M, an initial configuration v, and a measurable set of paths $W \subseteq V^\omega$, let $\mathsf{Prob}_{M_v}(W)$ denote the measure of W.

A set of plays $W \subseteq V^\omega$ is a *parity* condition if there is a parity priority function $\alpha : V \to \{0, \ldots, d\}$, with d as its *index*, such that a play $\pi = v_0, v_1, \ldots$ is in W iff $\min\{c \in \{0, \ldots, d\} \mid \exists^\infty i \,.\, \alpha(v_i) = c\}$ is even. A parity condition with $d = 1$ is a Büchi condition identified with the set $B = \alpha^{-1}(0)$. A parity condition with $d = 2$ and $\alpha^{-1}(0) = \emptyset$ is a co-Büchi condition identified with the set $C = \alpha^{-1}(1)$.

A *strategy* σ for Player 0 is $\sigma : V^* \cdot V_0 \to \mathcal{D}(V)$, such that $\sigma(w \cdot v)(v') > 0$ implies $(v, v') \in E$. A strategy π for Player 1 is defined similarly. A strategy is *pure* if it uses only pure distributions. Let w range over V^* and v over V. A strategy for Player 0 uses memory m if there is a domain M of size m with an initial value $m_0 \in M$ and two functions $\sigma_s : M \times V_0 \to \mathcal{D}(V)$ and $\sigma_u : M \times V \to M$ such that for $v \in V_0$ we have $\sigma(v) = \sigma_s(m_0, v_0)$ and $\sigma(w \cdot v) = \sigma_s(m_w, v)$, where $m_{v_0} = \sigma_m(v_0, m_0)$ and $m_{w \cdot v} = \sigma_m(m_w, v)$. Two strategies σ and π for both players and an initial configuration $v \in V$ induce a Markov chain $v(\sigma, \pi) = (S(v), (\emptyset, \emptyset, S(v)), E', \kappa')$, where $S(v) = \{v\} \cdot V^*$, $E' = \{(w, w \cdot v)\}$, and if $v \in V_0$ we have $\kappa'(wv) = \sigma(wv)$, if $v \in V_1$ we have $\kappa'(wv) = \pi(wv)$ and if $v \in V_p$ then for every $w \in V^*$ and $v' \in V$ we have $\kappa'(wv, wvv') = \kappa(v, v')$. We denote the set of strategies for Player 0 by Σ and the set of strategies for Player 1 by Π.

For a game G, an ω-regular set of plays W, and a configuration v, the value of W from v for Player 0, denoted $\mathsf{val}_0(W,v)$, and for Player 1, denoted $\mathsf{val}_1(W,v)$, are $\mathsf{val}_0(W,v) = \sup_{\sigma\in\Sigma}\inf_{\pi\in\Pi} \mathsf{Prob}_{v(\sigma,\pi)}(W)$ and $\mathsf{val}_1(W,v) = \sup_{\pi\in\Pi}\inf_{\sigma\in\Sigma} \left(1 - \mathsf{Prob}_{v(\sigma,\pi)}(W)\right)$.

We say that Player 0 wins W *surely* from v if $\exists\sigma \in \Sigma \,.\, \forall\pi \in \Pi \,.\, v(\sigma,\pi) \subseteq W$, where by $v(\sigma,\pi) \subseteq W$ we mean that *all* paths in $v(\sigma,\pi)$ are in W. We say that Player 0 wins W *almost surely* from v if $\exists\sigma \in \Sigma \,.\, \forall\pi \in \Pi \,.\, \mathsf{Prob}_{v(\sigma,\pi)}(W) = 1$. We say that Player 0 wins W *limit surely* from v if $\forall r < 1 \,.\, \exists\sigma \in \Sigma \,.\, \forall\pi \in \Pi \,.\, \mathsf{Prob}_{v(\sigma,\pi)}(W) \geq r$. In a given setup (e.g, almost-sure) if Player 0 cannot win we say that Player 1 wins. A strategy σ for Player 0 is optimal if $\mathsf{val}_0(W,v) = \inf_{\pi\in\Pi} \mathsf{Prob}_{v(\sigma,\pi)}(W)$. Optimality for Player 1 is defined similarly.

A game with condition W is *determined* if for every configuration v we have $\mathsf{val}_0(W,v) + \mathsf{val}_1(W,v) = 1$.

3 Sure-Almost-Sure MDPs

Berthon et al. considered the case of MDPs with two parity conditions and finding a strategy that has to satisfy one of the conditions surely and satisfy a given probability threshold with respect to the other [5]. Here we consider the case that the second condition has to hold with probability 1. We consider winning conditions composed of two parity conditions. The goal of Player 0 is to have one strategy such that she can win *surely* for the *sure* winning condition and *almost-surely* for the *almost-sure* winning condition. The authors of [5] show that optimal strategies exist in this case and that it can be decided whether Player 0 can win. Here we revisit their claim that Player 0 may need infinite memory in order to win in such an MDP. We then show that checking whether she can win using a finite-memory strategy is simpler than deciding if there is a general winning strategy.

Given a set of configurations V, a sure-almost-sure winning condition is $\mathcal{W} = (W_s, W_{as})$, where $W_s \subseteq V^\omega$ and $W_{as} \subseteq V^\omega$ are two parity winning conditions. A sure-almost-sure (SAS) MDP is $G = (V,(V_0,V_p),E,\kappa,\mathcal{W})$, where all components are as before and where $\mathcal{W}$ is a sure-almost-sure winning condition. Strategies for Player 0 are defined as before. We say that Player 0 wins from configuration v if the same strategy σ is winning surely with respect to W_s and almost-surely with respect to W_{as}.

▶ **Theorem 1** ([5]). *In a finite SAS parity MDP deciding whether a configuration v is winning for Player 0 is in NP ∩ co-NP. Furthermore, there exists an optimal infinite-state strategy for the joint goal.*

There exist SAS MDPs where Player 0 wins but not with finite-memory.

▶ **Theorem 2** ([5]). *For SAS MDPs finite-memory strategies do not capture winning.*

In the proof (in [18]) we revisit the MDP in Figure 1 (due to [5]) and repeat their argument showing that there is an infinite-memory strategy that can win both the sure (visit $\{l,r\}$ infinitely often) and almost-sure (visit $\{r\}$ finitely often) winning conditions. Intuitively, longer and longer attempts to reach l at c ensure infinitely many visits to $\{l,r\}$ and finitely many visits to r with probability 1. We present a detailed proof that every finite-memory strategy winning almost-surely is losing with respect to the sure winning condition.

The following theorem is proven by a chain of reductions (see proof in [18]). First, reduce the winning in an SAS MDP to the winning in an SAS MDP where the almost-sure winning condition is a Büchi condition. Second, we reduce the winning in an SAS MDP with a Büchi almost-sure winning condition to the winning in a (non-stochastic) game with the winning

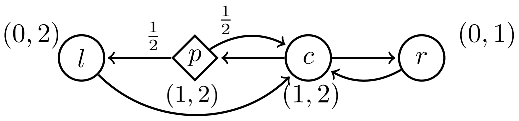


Figure 1 An SAS MDP where Player 0 requires infinite memory to win [5]. Configuration p is probabilistic and configurations l, c, and r are Player 0 configurations. The parity is induced by the following priorities $\alpha_s(l) = \alpha_s(r) = 0$, $\alpha_s(p) = \alpha_s(c) = 1$, and $\alpha_{as}(r) = 1$ and $\alpha_{as}(l) = \alpha_{as}(c) = \alpha_{as}(p) = 2$.

condition a conjunction of parity and Büchi. This is a special case of Theorem 8. Third, we reduce the winning in a game with a winning condition that is the conjunciton of parity and Büchi to winning in a parity game. Formally, we have the following.

▶ **Theorem 3.** *In order to decide whether it is possible to win an SAS MDP with n locations and indices d_s and d_{as} with finite memory it is sufficient to solve a (non-stochastic) parity game with $O(n \cdot d_s \cdot d_{as})$ configurations and index d_s. Furthermore, d_s is a bound on the size of the required memory in case of a win.*

▶ **Corollary 4.** *Consider an SAS MDP with n configurations, sure winning condition of index d_s, and almost-sure winning condition of index d_{as}. Checking whether Player 0 can win with finite-memory can be computed in quasi-polynomial time. In case that $d_s \leq \log n$ it can be decided in polynomial time.*

Proof. This is a direct result of Theorem 3 and the quasi-polynomial algorithm for solving parity games in [8, 30]. ◀

4 Sure-Almost-Sure Parity Games

We now turn our attention to sure-almost-sure parity games.

A sure-almost-sure (SAS) parity game is $G = (V, (V_0, V_1, V_p), E, \kappa, \mathcal{W})$, where all components are as before and $\mathcal{W}$ consists of *two* parity conditions $W_s \subseteq V^\omega$ and $W_{as} \subseteq V^\omega$. Strategies and the resulting Markov chains are as before. We say that Player 0 wins G from configuration v if she has a strategy σ such that for every strategy π of Player 1 we have $v(\sigma, \pi) \subseteq W_s$ and $\mathsf{Prob}_{v(\sigma,\pi)}(W_{as}) = 1$. That is, Player 0 has to win for sure (on all paths) with respect to W_s and with probability 1 with respect to W_{as}. Otherwise, Player 1 wins.

4.1 Determinacy

We start by showing that SAS parity games are determined.

▶ **Theorem 5.** *SAS parity games are determined.*

In the proof (in [18]) we use a reduction similar to Martin's proof that Blackwell games are determined [31]. We reduce SAS games to turn-based two-player games in a way that preserves winning.

4.2 General Winning

We show that determining whether Player 0 has a (general) winning strategy in an SAS parity game is co-NP-complete and that for Player 1 memoryless strategies are sufficient and that deciding her winning is NP-complete.

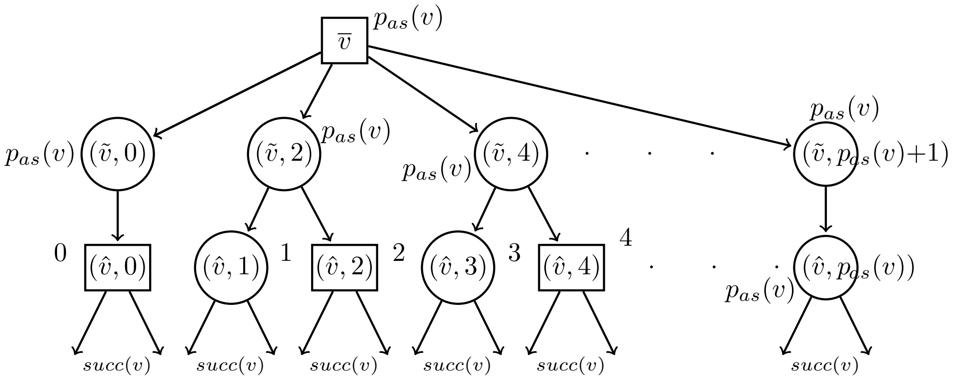


Figure 2 Gadget replacing probabilistic configurations for a configuration with odd parity.

▶ **Theorem 6.** *In an SAS parity game Player 1 has optimal memoryless strategies.*

The proof (in [18]) is by an inductive argument over the number of configurations of Player 1 (similar to that done in [28, 27, 10]).

▶ **Corollary 7.** *Consider an SAS parity game. Deciding whether Player 1 wins is NP-complete and whether Player 0 wins is co-NP-complete.*

Proof. Consider the case of Player 1. The optimal strategy for Player 1 is memoryless. Fixing Player 1's strategy in the game results in an SAS MDP. According to Theorem 1, the winning for Player 0 in SAS MDPs is in NP∩co-NP. The NP algorithm is as follows: it guess the memoryless strategy of Player 1 in the game, and the required polynomial witness of the SAS MDP, and use the polynomial-time verification procedure of the SAS MDP given the witness.[1] Hardness is by considering SAS games with no stochastic configurations [13].

Consider the case of Player 0. Membership in co-NP follows from dualizing the previous argument about membership in NP and determinacy. Hardness follows from considering SAS games with no stochastic configurations [13]. ◀

4.3 Winning with Finite Memory

We show that in order to check whether Player 0 can win with finite memory it is enough to use the standard reduction from almost-sure winning in two-player stochastic parity games to sure winning in two-player parity games [15].

▶ **Theorem 8.** *In a finite SAS parity game with n locations and d_{as} almost-sure index deciding whether a node v is winning for Player 0 with finite memory can be decided by a reduction to a two-player (non-stochastic) game with $O(n \cdot d_{as})$ locations, where the winning condition is the intersection of two parity conditions of indices d_s and d_{as}.*

The proof has the following steps: Given an SAS parity game G, we construct a non-stochastic game G' with conjunction of two objectives with a mapping between configurations of G and G'. We show that we can win from a configuration in G if and only if we can

[1] Note that we do not require a general NP algorithm with NP ∩co-NP oracle (such algorithms can make polynomially many queries to the oracle, as well as adaptive queries where queries can depend on answers of previous queries). Instead we have a NP algorithm with a single query to a NP∩co-NP oracle, and outputs the answer of the oracle, and thus can be implemented by a single NP algorithm.

win from its mapped configuration in G'. In one direction, we show that given winning strategy in G', we can construct winning strategy in G (from the mapped configurations). The construction of the winning strategy is based on the translation of a ranking function in G' to an almost-sure ranking function in G. Such a ranking function ensures winning the SAS objective in G. In the other direction, we show that given a winning strategy in G, we can construct a winning strategy in G' (from the mapped configurations). As before, the construction of the winning strategy is based on the translation of a ranking function in G to a ranking function in G'.

Proof. Let $G = (V, (V_0, V_1, V_p), E, \kappa, \mathcal{W})$. Let $p_{as} : V \to [0..d_{as}]$ be the parity priority function that induces W_{as} and $p_s : V \to [0..d_s]$ be the parity priority function that induces W_s. Without loss of generality assume that both d_s and d_{as} are even.

Given G we construct the game G' where every configuration $v \in V_p$ is replaced by the gadget in Figure 2. That is, $G' = (V', (V'_0, V'_1), E', \kappa', \mathcal{W}')$, with the following components:

- $V'_0 = V_0 \cup \left\{ (\tilde{v}, 2i), (\hat{v}, 2j-1) \;\middle|\; \begin{array}{l} v \in V_p,\ 2i \in [0..p_{as}(v)+1], \\ \text{and } 2j-1 \in [1..p_{as}(v)] \end{array} \right\}$
- $V'_1 = V_1 \cup \{\overline{v}, (\hat{v}, 2i) \mid v \in V_p \text{ and } 2i \in [0..p_{as}(v)]\}$
- $E' = \{(v,w) \mid (v,w) \in E \cap (V_0 \cup V_1)^2\} \quad \cup \quad \{(v, \overline{w}) \mid (v,w) \in E \cap (V_0 \cup V_1) \times V_p\} \quad \cup$ $\{((\hat{v}, j), w) \mid (v,w) \in E \cap V_p \times (V_0 \cup V_1)\} \quad \cup \quad \{((\hat{v}, j), \overline{w}) \mid (v,w) \in E \cap V_p^2\} \quad \cup$ $\{(\overline{v}, (\tilde{v}, 2i)) \mid v \in V_p\} \quad \cup \quad \{((\tilde{v}, 2i), (\hat{v}, j)) \mid v \in V_p \text{ and } j \in \{2i, 2i-1\}\}$
- $\mathcal{W}' = W'_s \cap W'_{as}$, where W'_s and W'_{as} are the parity winning sets that are induced by the following priority functions.

$$p'_{as}(t) \quad = \quad \begin{cases} p_{as}(t) & t \in V_0 \cup V_1 \\ p_{as}(v) & t \in \{\overline{v}, (\tilde{v}, 2i)\} \\ j & t = (\hat{v}, j) \end{cases} \qquad p'_s(t) \quad = \quad \begin{cases} p_s(t) & t \in V_0 \cup V_1 \\ p_s(v) & t \in \{\overline{v}, (\tilde{v}, 2i), (\hat{v}, j)\} \end{cases}$$

We show that Player 0 surely wins from a configuration $v \in V_0 \cup V_1$ in G' iff she wins from v in G with a pure finite-memory strategy and she wins from $\overline{v} \in V'$ in G' iff she wins from v in G with a pure finite-memory strategy.

The game G' is a linear game whose winning condition (for Player 0) is an intersection of two parity conditions. It is known that such games are determined and that the winning sets can be computed in NP $\cap$ co-NP [13]. Indeed, the winning condition for Player 0 can be expressed as a Streett condition, and hence her winning can be decided in co-NP. The winning condition for Player 1 can be expressed as a Rabin condition, and hence her winning can be decided in NP. It follows that V' can be partitioned to W'_0 and W'_1, the winning regions of Player 0 and Player 1, respectively. Furthermore, Player 0 has a pure finite-memory winning strategy for her from every configuration in W'_0 and Player 1 has a pure memoryless winning strategy for her from every configuration in W'_1. Let σ'_0 denote the winning strategy for Player 0 on W'_0 and π'_1 denote the winning strategy for Player 1 on W'_1. Let M be the memory domain used by σ'_0. As σ'_0 is pure, we can think about it as $\sigma'_0 \subseteq V' \times M \to V' \times M$, where for every $m \in M$ and $v \in V'_0$ there is a unique $w \in V$ and $m' \in M$ such that $((v,m),(w,m')) \in \sigma'$ and for every $m \in M$ and $v \in V'_1$ and w such that $(v,w) \in E'$ there is a unique m' such that $((v,m),(w,m')) \in \sigma'_0$. We freely say σ'_0 chooses v' from (v,m) for the unique v' such that $(v,m,v',m') \in \sigma'_0$ for some m' and σ'_0 updates the memory to m'. Similarly, a pure strategy in G can be described as $\sigma \subseteq (V \times M)^2$ where stochastic configurations are handled like Player 1 configuration in term of memory update for all successors as above. By abuse of notation we refer to the successor of a configuration v in G' and mean either w or $\overline{w}$ according to the context.

$\Leftarrow$ We show that every configuration $v \in W_0'$ that is winning for Player 0 in G' is in the winning region W_0 of Player 0 in G. Consider the strategy $\sigma_0' \subseteq (V' \times M)^2$. We construct a winning strategy $\sigma_0 \subseteq (V \times M)^2$, induced by σ_0' as follows:

- For a configuration-memory (cm) pair $(v, m) \in V_0 \times M$ there is a unique cm pair (v', m') such that $(v, m, v', m') \in \sigma_0'$. We set $(v, m, v', m') \in \sigma_0$.
- For a cm pair $(v, m) \in V_1 \times M$ and for every successor w of v there is a unique memory value m' such that $(v, m, w, m') \in \sigma_0'$. We set $(v, m, w, m') \in \sigma_0$.
- Consider a cm pair $(v, m) \in V_p \times M$. As $\overline{v}$ is a Player 1 configuration in G', for every configuration $(\tilde{v}, 2i)$ there is a uniqye m' such that $(v, m, (\tilde{v}, 2i), m') \in \sigma_0'$.
  - * If for some i we have that the choice from $(\tilde{v}, 2i)$ according to σ_0' is $(\hat{v}, 2i-1)$. Then, let i_0 be the minimal such i and let w_0 be the successor of v such that the choice of σ_0' from $(\hat{v}, 2i_0 - 1)$ is w_0. We update in σ_0 the tuple (v, m, w_0, m'), where m' is the memory resulting from taking the path $\overline{v}$, $(\tilde{v}, 2i_0)$, $(\hat{v}, 2i_0 - 1)$, w_0 in G' based on σ_0'. We update in σ_0 the tuple $(v, m, w', m_{w'})$ for $w' \neq w_0$, where $m_{w'}$ is the memory resulting from taking the path $\overline{v}$, $(\tilde{v}, 2i_0 - 2)$, $(\hat{v}, 2i_0 - 2)$, w'. Notice that as i_0 is chosen to be the minimal the choice from $(\tilde{v}, 2i_0 - 2)$ to $(\hat{v}, 2i_0 - 2)$ is compatible with σ_0', where $2i_0 - 2$ could be 0.
  - * If for all i we have that the choice from $(\tilde{v}, 2i)$ according to σ_0' is $(\hat{v}, 2i)$. Then, for every w successor of v we update in σ_0 the tuple (v, m, w, m'), where m' is the memory resulting from taking the path $\overline{v}$, $(\tilde{v}, p_{as}(v))$, $(\hat{v}, p_{as})$, w.

  Notice that if $p_{as}(v)$ is odd then the first case always holds as the only successor of $(\tilde{v}, p_{as}(v) + 1)$ is $(\hat{v}, p_{as}(v))$.

The resulting strategy σ_0 includes no further decisions for Player 0. Consider the winning condition W_s. Every path in G that is consistent with σ_0 (with proper memory updates) corresponds to a path in G' that is consistent with σ_0' (with the same memory updates) and agrees on the parities of all configurations according to p_s. Indeed, every configuration of the form $(\tilde{v}, 2i)$ or $(\hat{v}, j)$ in G' has the same priority according to p_s as $\overline{v}$ (and v in G). As every path consistent with σ_0' is winning according to W_s' then every path in G consistent with σ is winning according to W_s.

We turn our attention to consider only the parity condition p_{as} in both G' and G. We think about G' as a parity game with the winning condition W_{as}' and about G as a stochastic parity game with the winning condition W_{as}. As σ_0' is winning, all paths in G' (with proper memory updates) are winning for Player 0 according to W_{as}'.

We recall some definitions and results from [15]. For $k \leq d_{as}$, let $\underline{k}$ denote k if k is odd and $k - 1$ if k is even. A *parity ranking* for Player 0 is $\vec{r} : V' \times M \to [n]^{d_{as}/2} \cup \{\infty\}$ for some $n \in \mathbb{N}$, where $[n]$ denotes $\{0, \ldots, n\}$. For a configuration v, Let $\vec{r}(v) = (r_1, \ldots, r_d)$ and $\vec{r}(v') = (r_1', \ldots, r_d')$, where $d = d_{as}/2$. For v, we denote by $\vec{r}^k(v)$ the prefix $(r_1, r_3, \ldots, r_{\underline{k}})$ of $\vec{r}(v)$. We write $\vec{r}(v) \leq_k \vec{r}(v')$ if the prefix $(r_1, \ldots r_{\underline{k}})$ is at most $(r_1', \ldots, r_{\underline{k}}')$ according to the lexicographic ordering. Similarly, we write $\vec{r}(v) <_k \vec{r}(v')$ if $(r_1, \ldots r_{\underline{k}})$ is less than $(r_1', \ldots, r_{\underline{k}}')$ according to the lexicographic ordering.

A parity ranking is good if (i) for every vertex $v \in V_0$ and memory $m \in M$ there is a vertex $w \in \mathit{succ}(v)$ and $m' \in M$ such that $\vec{r}(w, m') \leq_{p(v)} \vec{r}(v, m)$ and if $p(v)$ is odd then $\vec{r}(w, m') <_{p(v)} \vec{r}(v, m)$ and (ii) for every vertex $v \in V_1$, memory $m \in M$, and vertex $w \in \mathit{succ}(v)$ it holds that there is a $m' \in M$ such that $\vec{r}(w, m') \leq_{p(v)} \vec{r}(v, m)$ and if $p(v)$ is odd then $\vec{r}(w, m') <_{p(v)} \vec{r}(v, m)$. It is well known that in a parity game (here G' combined with the strategy σ_0') there is a *good* parity ranking such that for every $v \in W_0'$ and memory $m \in M$ we have $\vec{r}(v, m) \neq \infty$ [29]. Let $\vec{r}$ be the good parity ranking for G'. Consider the same ranking for G with the same memory M. For a cm pair $(v,m) \in V_p \times M$, we write $\mathsf{Prob}_{v,m}(\vec{r}_{\leq_k})$ for the probability (according to κ) of successors w of v such that for some memory values m_w we have $\vec{r}(w,m_w) \leq_k \vec{r}(v,m)$ and $\mathsf{Prob}_{v,m}(\vec{r}_{<_k})$ for the probability of successors w of v such that for some memory values m_w we have $\vec{r}(w,m_w) <_k \vec{r}(v,m)$.

▶ **Definition 9** (Almost-sure ranking [14]). *A ranking function $\vec{r} : V \times M \to [n]^{d_{as}/2} \cup \{\infty\}$ for Player 0 is an almost-sure ranking if there is an $\epsilon \geq 0$ such that for every pair (v,m) with $r(v,m) \neq \infty$, the following conditions hold:*

- *If $v \in V_0$ there exists a successor w and memory m' such that $\vec{r}(w,m') \leq_{p(v)} \vec{r}(v,m)$ and if $p(v)$ is odd then $\vec{r}(w,m') <_{p(v)} \vec{r}(v,m)$.*
- *If $v \in V_1$ then for every successor w of v there is a memory m' such that $\vec{r}(w,m') \leq_{p(v)} \vec{r}(v,m)$ and if $p(v)$ is odd then $\vec{r}(w,m') <_{p(v)} \vec{r}(v,m)$.*
- *If $v \in V_p$ and $p(v)$ is even then either $\mathsf{Prob}_{v,m}(\vec{r}_{\leq_{p(v)-1}}) = 1$ or*
$$\bigvee_{j=2i+1\in[1..p(v)]} (\mathsf{Prob}_{v,m}(\vec{r}_{\leq_{j-2}}) = 1 \wedge \mathsf{Prob}_{v,m}(\vec{r}_{<_j}) \geq \epsilon)$$
- *If $v \in V_p$ and $p(v)$ is odd then* $\displaystyle\bigvee_{j=2i+1\in[1..p(v)]} \big(\mathsf{Prob}_{v,m}(\vec{r}_{\leq_{j-2}}) = 1 \wedge \mathsf{Prob}_{v,m}(\vec{r}_{<_j}) \geq \epsilon\big)$

▶ **Lemma 10** ([14]). *A stochastic parity game has an almost-sure ranking iff Player 0 can win for the parity objective with probability 1 from every configuration v such that for some m we have $\vec{r}(v,m) \neq \infty$.*

The following lemma specializes a similar lemma in [14] for our needs.

▶ **Lemma 11.** *The good ranking of G' with M induces an almost-sure ranking of G with M.*

Proof. Let ϵ be the minimal probability of a transition in G. As G is finite ϵ exists. For configurations in $V_0 \cup V_1$ the definitions of good parity ranking and almost-sure ranking coincide.

Consider a configuration $v \in V_p$ a memory $m \in M$ and the matching configuration $\overline{v}$. Let $p = p_{as}(v)$. Consider the pair (v,m) in $V \times M$ and $(\overline{v},m)$ in $V' \times M$. We consider the cases where p is even and when p is odd.

- Suppose that p is even. If there is some minimal i such that the choice of σ'_0 from $((\tilde{v},2i),m')$ in G' is $((\hat{v},2i-1),m'')$. Then, there is some $w \in succ(v)$ and some m''' such that $\vec{r}(w,m''') <_{2i-1} \vec{r}((\hat{v},2i-1),m'') \leq_p \vec{r}((\tilde{v},2i),m') \leq_p \vec{r}(\overline{v},m)$. It follows that $\mathsf{Prob}_{v,m}(\vec{r}_{<_{2i-1}}) \geq \epsilon$. Furthermore, as i is minimal it follows that $i \neq 0$ and that the choice of σ'_0 from $((\tilde{v},2i-2),n)$ is $((\hat{v},2i-2),n')$ and $(\tilde{v},2i-2)$ belongs to Player 1 in G'. Then, for every successor w of $(\hat{v},2i-2)$ and for every memory value n' there is a memory value n''' such that
$$\vec{r}(w,n''') \leq_{2i-2} \vec{r}((\hat{v},2i-2),n'') \leq_p ((\tilde{v},2i-2),n') \leq_p (\overline{v},m).$$
It follows that $\mathsf{Prob}_{v,m}(\vec{r}_{\leq_{2i-2}}) = 1$.
If there is no such i, then the choice of σ'_0 from $((\tilde{v},p),m')$ in G' is $((\hat{v},p),m'')$ and for every $w \in \mathsf{succ}(v)$ there is some m''' such that
$$\vec{r}(w,m''') \leq_p \vec{r}((\hat{v},p),m'') \leq_p \vec{r}((\tilde{v},p),m') \leq_p \vec{r}(\overline{v},m).$$
If follows that $\mathsf{Prob}_{v,m}(\vec{r}_{\leq_p}) = 1$.
- Suppose that p is odd. In this case there must be some minimal i such that the choice of σ'_0 from $((\tilde{v},2i),m')$ is $((\hat{v},2i-1),m'')$. We can proceed as above. ◀

As Player 0 has no further choices in G, it follows that the strategy σ_0 defined above is winning in G. That is, sure winning w.r.t. W_s and almost-sure winning w.r.t. W_{as}.

$\Rightarrow$ In the proof (in [18]) we show how to use a winning finite-memory strategy in G to induce a strategy in G' and use a ranking argument to show that this strategy is winning. ◀

▶ **Corollary 12.** *Consider an SAS turn-based stochastic parity game. Deciding whether Player 0 can win with finite-memory is co-NP-complete. Deciding whether Player 1 can win against finite-memory is NP-complete.*

Proof. Upper bounds follow from the reductions to Streett and Rabin winning conditions. Completeness follows from the case where the game has no stochastic configurations [13]. ◀

▶ Remark 13. The complexity established above in the case of finite-memory is the same as that established for the general case in Corollary 7. However, this reduction gives us a clear algorithmic approach to solve the case of finite-memory strategies. Indeed, in the general case, the proof of the NP upper bound requires enumeration of all memoryless strategies, and does not present an algorithmic approach, regardless of the indices of the different winning conditions. In contrast our reduction for the finite-memory case to non-stochastic games with conjunction of parity conditions and recent algorithmic results on non-stochastic games with ω-regular conditions of [8] imply the following:

- For the finite-memory case, we have a fixed parameter tractable algorithm that is polynomial in the number of the game configurations and exponential only in the indices to compute the SAS winning region.
- For the finite-memory case, if both indices are constant or logarithmic in the number of configurations, we have a polynomial time algorithm to compute the SAS winning region.

5 Sure-Limit-Sure Parity Games

In this section we extend our results to the case where the unsure goal is required to be met with limit-sure certainty, rather than almost-sure certainty.

Sure-limit-sure parity games. A *sure-limit-sure (SLS) parity game* is, as before, $G = (V, (V_0, V_1, V_p), E, \kappa, \mathcal{W})$. We denote the second winning condition with the subscript ls, i.e., W_{ls}. We say that Player 0 wins G from configuration v if she has a sequence of strategies $\sigma_i \in \Sigma$ such that for every i for every strategy π of Player 1 we have $v(\sigma_i, \pi) \subseteq W_s$ and $\mathsf{Prob}_{v(\sigma_i,\pi)}(W_{ls}) \geq 1 - \frac{1}{i}$. That is, Player 0 has a sequence of strategies that are sure winning (on all paths) with respect to W_s and ensure satisfaction probabilities approaching 1 with respect to W_{ls}.

5.1 Limit-Sure vs Almost-Sure

In MDPs and stochastic turn-based games with parity conditions almost-sure and limit-sure winning coincide [9]. In contrast to the above result we present an example MDP where in addition to surely satisfying one parity condition limit-sure winning with another parity condition can be ensured, but almost-sure winning cannot be ensured. In other words, in conjunction with sure winning, limit-sure winning does not coincide with almost-sure winning even for MDPs. Such a result was established in [5] for MDPs with infinite-memory strategies. We show the same holds for finite-memory strategies.

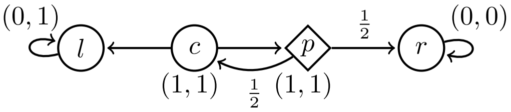


Figure 3 An MDP where Player 0 can ensure sure winning and win limit-surely but cannot win almost-surely. Configuration p is probabilistic and configurations l, c, and r are Player 0 configurations. The winning conditions are induced by the following priorities $\alpha_s(l) = \alpha_s(r) = 0$, $\alpha_s(p) = \alpha_s(c) = 1$, and $\alpha_{ls}(r) = 0$ and $\alpha_{ls}(l) = \alpha_{ls}(c) = \alpha_{ls}(p) = 1$.

▶ **Theorem 14.** *While satisfying one parity condition surely, the almost-sure winning set for another parity condition is a strict subset of limit-sure winning set, even in the context of MDPs with finite-memory strategies.*

Proof. Consider the MDP in Figure 3. Clearly, Player 0 wins surely with respect to both parity conditions in configuration r and Player 0 cannot win the condition W_{ls} on l. In order to win W_s the cycle between p and c has to be taken finitely often. Then, the edge from c to l must be taken eventually. However, l is a sink that is losing with respect to W_{ls}. It follows, that Player 0 cannot win almost-surely with respect to W_{ls} while winning surely with respect to W_s.

On the other hand, for every $\epsilon > 0$ there is a finite-memory strategy that is sure winning with respect to W_s and wins with probability at least $1 - \epsilon$ with respect to W_{ls}. Indeed, Player 0 has to choose the edge from c to p at least N times, where N is large enough such that $\frac{1}{2^N} < \epsilon$, and then choose the edge from c to l. Then, Player 0 wins surely with respect to W_s (every play eventually reaches either l or r) and with probability more than $1 - \epsilon$ with respect to W_{ls}.

To summarize, Player 0 wins surely w.r.t. W_s and limit-surely w.r.t. W_{ls} from both c and p but cannot win almost-surely w.r.t. W_{ls} from c and p. ◀

5.2 Solving SLS MDPs and Games

We first note that Player 1 has optimal memoryless strategies similar to the SAS case. The proof (in [18]) reuses the proof of Theorem 6.

▶ **Theorem 15.** *In an SLS parity game Player 1 has optimal memoryless strategies.*

SLS MDPs. We now present the solution to winning in SLS MDPs. Given an SLS MDP G with winning conditions W_s and W_{ls}, we call the *induced* SAS MDP the MDP with winning conditions W_s and W_{ls}, where the latter is interpreted as an almost-sure winning condition. We use the induced SAS MDP in the solution of the SLS MDP. The memory used in the SLS part has to match the memory used for winning in the SAS part. That is, if Player 0 is restricted to finite-memory in the SLS part of the game she has to consider finite-memory strategies in the induced SAS MDP.

▶ **Theorem 16.** *In a finite SLS parity MDP deciding whether a node v is winning for Player 0 can be reduced to the limit-sure reachability while maintaining sure-parity. The target of the limit-sure reachability is the winning region of the induced SAS partiy MDP.*

Proof. *SAS winning region A.* Consider an MDP $G = (V, (V_0, V_p), E, \kappa, \mathcal{W})$, where $\mathcal{W} = (W_s, W_{ls})$. Consider G as an SAS MDP and compute the set of configurations from which Player 0 can win G. Let $A \subseteq V$ denote this winning region and $B = V \setminus A$ be the complement region. Clearly, A is *closed* under probabilistic moves. That is, if $v \in V_p \cap A$ then for every v' such that $(v, v') \in E$ we have $v' \in A$. Furthermore, under Player 0's winning strategy, Player 0 does not use edges going back from A to B. It follows that we can consider A as a sink in G.

Reduction to limit-sure reachability. We present the argument for finite-memory strategies for Player 0, and the argument for infinite-memory strategies is similar. Consider an arbitrary finite-memory strategy $\sigma \in \Sigma$, and consider the Markov chain that is the result of restricting Player 0 moves according to σ.

- *Bottom SCC property.* Let S be a bottom SCC (SCC that is only reachable from itself) that intersects with B in the Markov chain. As explained above, it cannot be the case that this SCC intersects A (since we consider A as sink due to the closed property). Thus the SCC S must be contained in B. Thus, either S must be losing according to W_s or the minimal parity in S according to W_{ls} is odd, as otherwise in the region S Player 0 ensures sure winning wrt W_s and almost-sure winning wrt W_{ls}, which means that S belongs to the SAS winning region A. This contradicts that S is contained in B.
- *Reachability to A.* In a Markov chain bottom SCCs are reached with probability 1, and from the above item it follows that the probability to satisfy the W_{ls} goal along with ensuring W_s while reaching bottom SCCs in B is zero. Hence, the probability to satisfy W_{ls} along with ensuring W_s is at most the probability to reach A. On the other hand, after reaching A, the SAS goal can be ensured by switching to an appropriate SAS strategy in the winning region A, which implies that the SLS goal is ensured. Hence it follows that the SLS problem reduces to limit-sure reachability to A, while ensuring the sure parity condition W_s. ◀

▶ Remark 17. Note that for finite-memory strategies the argument above is based on bottom SCCs. The SAS region for MDPs wrt to infinite-memory strategies is achieved by characterizing certain strongly connected components (called Ultra-good end-components [5, Definition 5]), and hence a similar argument as above also works for infinite-memory strategies to show that SLS for infinite-memory strategies for two parity conditions reduces to limit-sure reachability to the SAS region while ensuring the sure parity condition (however, in this case the SAS region has to be computed for infinite-memory strategies).

Limit-sure reachability and sure parity in games. We consider the problem of Player 0 ensuring limit-sure reachability to target set A while preserving sure parity. We present the solution for games (which subsumes the case of MDPs).

▶ **Theorem 18.** *Consider an SLS Game, where the limit-sure condition is to reach a target set A that is also winning for the sure condition. Player 0's winning region is the limit-sure reachability region to A within the winning region of the sure parity condition.*

In one direction, in the limit-sure reachability to A within the sure winning region, the limit-sure reachability strategy can be played to enforce high probability of winning for the limit-sure winning condition and then revert to the sure-winning strategy. The combination delivers an arbitrarily high probability of reaching A as well as sure winning. In the other direction, a strategy that wins limit-sure reachability to A and sure-winning with respect to the sure condition is clearly restricted to the sure-winning region. At the same time, it ensures limit-sure reachability to A. Hence, the analysis of such games is simplified into two steps; first compute the sure winning region for the sure objective, and in this subgame only consider reachability to the limit-sure target set.

Proof. WLOG we replace the region A by a single configuration t with a self loop and an even priority with respect to W_s. Consider an SLS game, with a configuration t of sink target state, such that the limit-sure goal is to reach t, and t has even priority with respect to W_s. We now present solution to this limit-sure reachability with sure parity problem. The computational steps are as follows:

- First, compute the sure winning region w.r.t the parity condition in the game. Let X be this winning region. Note that $t \in X$ as t is a sink state with even priority for W_s.
- Second, restrict the game to X and compute limit-sure reachability region to t, and let the region be Y. Note that the game restricted to X is a turn-based stochastic game where almost-sure and limit-sure reachability coincide.

Let us denote by Z the desired winning region (i.e., from where sure parity can be ensured along with limit-sure reachability to t). We argue that Y computes the desired winning region Z as follows:

- First, note that since the sure parity condition W_s must be ensured, the sure winning region X must never be left. Thus without loss of generality, we can restrict the game to X. By definition Y is the region in X to ensure limit-sure reachability to t. As Z ensures both limit-sure reachability to t as well as sure parity, it follows that Z is a subset of Y.
- Second, for any $\epsilon > 0$, there is a strategy in Y to ensure that t is reached with probability at least $1 - \epsilon$ within N_ϵ steps staying in X (since in the subgame restricted to X, almost-sure reachability to t can be ensured). Consider a strategy that plays the above strategy for N_ϵ steps, and if t is not reached, then switches to a sure winning strategy for W_s (such a strategy exists since X is never left, and parity conditions are independent of finite prefixes). It follows that from Y both limit-sure reachability to t as well as sure parity condition W_s can be ensured. Hence $Y \subseteq Z$.

Thus, $Y = Z$ as required. ◀

▶ **Corollary 19.** *Consider an SLS turn-based stochastic parity game. Deciding whether Player 0 wins is co-NP-complete. Deciding whether Player 1 wins is NP-complete. Consider an SLS turn-based MDP with n locations and indices d_s and d_{ls}. Checking whether Player 0 can win with finite-memory can be computed in quasi-polynomial time. In case that $d_s \leq \log n$ it can be decided in polynomial time.*

Proof. It follows from above that to solve SLS MDPs, the following computation steps are sufficient: (a) solve SAS MDP, (b) compute sure winning region for parity condition, and (c) compute almost-sure (=limit-sure) reachability in MDPs. The second step is a special case of the first step, and the third step can be achieved in polynomial time [12, 19]. Hence it follows that all the complexity and algorithmic upper bounds we established for the SAS MDPs carry over to SLS MDPs. For games, since Player 1 has memoryless optimal strategies (Theorem 15) and the complexity of SAS MDPs and SLS MDPs coincide, the complexity upper bounds for SAS games carry over to SLS games. Finally, since the complexity lower bound results for SAS parity games follow from games with no stochastic transitions, they apply to SLS parity games as well. ◀

6 Conclusions and Future Work

In this work we consider MDPs and turn-based stochastic games with two parity winning conditions, with combinations of qualitative winning criteria. In particular, we study the case where one winning condition must be satisfied surely, and the other almost-surely (or limit-surely). We present results for MDPs with finite-memory strategies, and turn-based stochastic games with finite-memory and infinite-memory strategies. Our results establish complexity results, as well as algorithmic results for finite-memory strategies by reduction to non-stochastic games. Some interesting directions for future work are as follows. First, while our results establish algorithmic results for finite-memory strategies, whether similar results can be established for infinite-memory strategies is an interesting open question. Second, the study of the synthesis problem for turn-based stochastic games with combinations of quantitative objectives is another interesting direction of future work. If we consider more than two conjuncts with only two types, i.e., sure and almost-sure, or sure and limit-sure, then solution of the game reduces to a conjunction of two conditions. The problem of conjunctions with more than two types and general Boolean combinations of winning conditions are interesting directions for future work.

References

1 S. Almagor, O. Kupferman, and Y. Velner. Minimizing Expected Cost Under Hard Boolean Constraints, with Applications to Quantitative Synthesis. In *27th International Conference on Concurrency Theory*, volume 59 of *LIPIcs*, pages 9:1–9:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

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

3 C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

4 N. Basset, M.Z. Kwiatkowska, U. Topcu, and C. Wiltsche. Strategy Synthesis for Stochastic Games with Multiple Long-Run Objectives. In *Proc. 21st Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 9035 of *Lecture Notes in Computer Science*, pages 256–271. Springer, 2015.

5 R. Berthon, M. Randour, and J.-F. Raskin. Threshold Constraints with Guarantees for Parity Objectives in Markov Decision Processes. In *44th International Colloquium on Automata, Languages, and Programming*, volume 80 of *LIPIcs*, pages 121:1–121:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

6 T. Brázdil, V. Brozek, K. Chatterjee, V. Forejt, and A. Kucera. Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes. In *Proc. 26rd IEEE Symp. on Logic in Computer Science*, pages 33–42. IEEE Computer Society Press, 2011.

7 V. Bruyère, E. Filiot, M. Randour, and J.-F. Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. *Inf. Comput.*, 254:259–295, 2017.

8 C. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding Parity Games in Quasipolynomial Time. In *Proc. 49th ACM Symp. on Theory of Computing*, pages 252–263. ACM Press, 2017.

9 K. Chatterjee. *Stochastic ω-regular Games*. PhD thesis, University of California at Berkeley, 2007.

10 K. Chatterjee and L. Doyen. Perfect-Information Stochastic Games with Generalized Mean-Payoff Objectives. In *Proc. 31st IEEE Symp. on Logic in Computer Science*, pages 247–256. ACM Press, 2016.

11 K. Chatterjee, L. Doyen, M. Randour, and J.-F. Raskin. Looking at mean-payoff and total-payoff through windows. *Information and Computation*, 242:25–52, 2015.

12 K. Chatterjee and M. Henzinger. Faster and Dynamic Algorithms For Maximal End-Component Decomposition And Related Graph Problems In Probabilistic Verification. In *SODA'11*. SIAM, 2011.

13 K. Chatterjee, T.A. Henzinger, and N. Piterman. Generalized Parity Games. In *Proc. 10th Int. Conf. on Foundations of Software Science and Computation Structures*, volume 4423 of *Lecture Notes in Computer Science*, pages 153–167, Braga, Porgugal, 2007. Springer.

14 K. Chatterjee, M. Jurdziński, and T.A. Henzinger. Simple Stochastic Parity Games. In *Proc. 12th Annual Conf. of the European Association for Computer Science Logic*, Lecture Notes in Computer Science, pages 100–113. Springer, 2003.

15 K. Chatterjee, M. Jurdziński, and T.A. Henzinger. Quantitative Stochastic Parity Games. In *Symposium on Discrete Algorithms*, pages 114–123. SIAM, 2004.

16 K. Chatterjee, Z. Komárková, and J. Kretínský. Unifying Two Views on Multiple Mean-Payoff Objectives in Markov Decision Processes. In *Proc. 30th IEEE Symp. on Logic in Computer Science*, pages 244–256. IEEE Computer Society Press, 2015.

17 K. Chatterjee, R. Majumdar, and T.A. Henzinger. Markov Decision Processes with multiple objectives. In *Proc. 23rd Symp. on Theoretical Aspects of Computer Science*, volume 3884 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2006.

18 K. Chatterjee and N. Piterman. Combinations of Qualitative Winning for Stochastic Parity Games. Technical report, arXiv, 2018. `arXiv:1804.03453`.

19 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.

20 T. Chen, V. Forejt, M.Z. Kwiatkowska, A. Simaitis, and C. Wiltsche. On Stochastic Games with Multiple Objectives. In *38th Int. Symp. on Mathematical Foundations of Computer Science*, volume 8087 of *Lecture Notes in Computer Science*, pages 266–277. Springer, 2013.

21 A. Church. Logic, arithmetics, and automata. In *Proc. Int. Congress of Mathematicians, 1962*, pages 23–35. Institut Mittag-Leffler, 1963.

22 A. Condon. The complexity of stochastic games. *Information and Computation*, 96:203–224, 1992.

23 C. Courcoubetis and M. Yannakakis. The Complexity of Probabilistic Verification. *jacm*, 42(4):857–907, 1995.

24 K. Etessami, M.Z. Kwiatkowska, M.Y. Vardi, and M. Yannakakis. Multi-Objective Model Checking of Markov Decision Processes. *Logical Methods in Computer Science*, 4(4), 2008.

25 Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes*. Springer, 1996.

26 V. Forejt, M.Z. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Quantitative Multi-objective Verification for Probabilistic Systems. In *Proc. 17th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 6605 of *Lecture Notes in Computer Science*, pages 112–127. Springer, 2011.

27 H. Gimbert and E. Kelmendi. Two-Player Perfect-Information Shift-Invariant Submixing Stochastic Games Are Half-Positional. *CoRR*, abs/1401.6575, 2014. `arXiv:1401.6575`.

28 H. Gimbert and W. Zielonka. Games Where You Can Play Optimally Without Any Memory. In *16th Int. Conf. on Concurrency Theory*, volume 3653 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2005.

29 M. Jurdziński. Small Progress Measures for Solving Parity Games. In *Proc. 17th Symp. on Theoretical Aspects of Computer Science*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301, Lille, France, 2000. Springer.

30 M. Jurdzinski and R. Lazic. Succinct progress measures for solving parity games. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–9, Reykjavik, Iceland, 2017. IEEE Computer Society Press.

31 D.A. Martin. The Determinacy of Blackwell Games. *The Journal of Symbolic Logic*, 63(4):1565–1581, 1998.

32 A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190. ACM Press, 1989.

33 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1st edition, 1994.

34 M. Randour, J.-F. Raskin, and O. Sankur. Variations on the Stochastic Shortest Path Problem. In *Proc. 16th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 8931 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2015.

35 W. Thomas. Automata on infinite objects. *Handbook of Theoretical Computer Science*, pages 133–191, 1990.

36 Y. Velner, K. Chatterjee, L. Doyen, T.A. Henzinger, A. Rabinovich, and J.-F. Raskin. The complexity of multi-mean-payoff and multi-energy games. *Information and Computation*, 241:177–196, 2015.

Near-Linear Time Algorithms for Streett Objectives in Graphs and MDPs

Krishnendu Chatterjee
IST Austria, Klosterneuburg, Austria
krish.chat@ist.ac.at

Wolfgang Dvořák
Institute of Logic and Computation, TU Wien, Austria
dvorak@dbai.tuwien.ac.at

Monika Henzinger
Theory and Application of Algorithms, University of Vienna, Austria
monika.henzinger@univie.ac.at

Alexander Svozil
Theory and Application of Algorithms, University of Vienna, Austria
alexander.svozil@univie.ac.at

Abstract

The fundamental model-checking problem, given as input a model and a specification, asks for the algorithmic verification of whether the model satisfies the specification. Two classical models for reactive systems are graphs and Markov decision processes (MDPs). A basic specification formalism in the verification of reactive systems is the strong fairness (aka Streett) objective, where given different types of requests and corresponding grants, the requirement is that for each type, if the request event happens infinitely often, then the corresponding grant event must also happen infinitely often. All ω-regular objectives can be expressed as Streett objectives and hence they are canonical in verification. Consider graphs/MDPs with n vertices, m edges, and a Streett objectives with k pairs, and let b denote the size of the description of the Streett objective for the sets of requests and grants. The current best-known algorithm for the problem requires time $O(\min(n^2, m\sqrt{m \log n}) + b \log n)$. In this work we present randomized near-linear time algorithms, with expected running time $\widetilde{O}(m + b)$, where the $\widetilde{O}$ notation hides poly-log factors. Our randomized algorithms are near-linear in the size of the input, and hence optimal up to poly-log factors.

2012 ACM Subject Classification Mathematics of computing → Combinatorial algorithms; Software and its engineering → Formal software verification

Keywords and phrases model checking, graph games, Streett games

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.7

Funding A. S. is fully supported by the Vienna Science and Technology Fund (WWTF) through project ICT15-003. K.C. is supported by the Austrian Science Fund (FWF) NFN Grant No S11407-N23 (RiSE/SHiNE) and an ERC Starting grant (279307: Graph Games). For M.H the research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement no. 340506.

1 Introduction

In this work we present near-linear (hence near-optimal) randomized algorithms for the strong fairness verification in graphs and Markov decision processes (MDPs). In the fundamental model-checking problem, the input is a *model* and a *specification*, and the algorithmic verification problem is to check whether the model *satisfies* the specification. We first describe the models and the specifications we consider, then the notion of satisfaction, and then previous results followed by our contributions.

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 7; pp. 7:1–7:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Models: Graphs and MDPs. Graphs and Markov decision processes (MDPs) are two classical models of reactive systems. The states of a reactive system are represented by the vertices of a graph, the transitions of the system are represented by the edges and non-terminating trajectories of the system are represented as infinite paths of the graph. Graphs are a classical model for reactive systems with nondeterminism, and MDPs extend graphs with probabilistic transitions that represent reactive systems with both nondeterminism and uncertainty. Thus graphs and MDPs are the standard models of reactive systems with nondeterminism, and nondeterminism with stochastic aspects, respectively [15, 2]. Moreover MDPs are used as models for concurrent finite-state processes [16, 28] as well as probabilistic systems in open environments [26, 23, 17, 2].

Specification: Strong fairness (aka Streett) objectives. A basic and fundamental specification formalism in the analysis of reactive systems is the *strong fairness condition*. The strong fairness conditions (aka Streett objectives) consist of k types of requests and corresponding grants, and the requirement is that for each type if the request happens infinitely often, then the corresponding grant must also happen infinitely often. Beyond safety, reachability, and liveness objectives, the most standard properties that arise in the analysis of reactive systems are Streett objectives, and chapters of standard textbooks in verification are devoted to it (e.g., [15, Chapter 3.3], [24, Chapter 3], [1, Chapters 8, 10]). In addition, ω-regular objectives can be specified as Streett objectives, e.g., LTL formulas and non-deterministic ω-automata can be translated to deterministic Streett automata [25] and efficient translations have been an active research area [6, 18, 22]. Consequently, Streett objectives are a canonical class of objectives that arise in verification.

Satisfaction. The notions of satisfaction for graphs and MDPs are as follows: For graphs, the notion of satisfaction requires that there is a trajectory (infinite path) that belongs to the set of paths specified by the Streett objective. For MDPs the satisfaction requires that there is a strategy to resolve the nondeterminism such that the Streett objective is ensured almost-surely (with probability 1). Thus the algorithmic model-checking problem of graphs and MDPs with Streett objectives is a central problem in verification, and is at the heart of many state-of-the-art tools such as SPIN, NuSMV for graphs [20, 14], PRISM, LiQuor, Storm for MDPs [23, 13, 17].

Our contributions are related to the algorithmic complexity of graphs and MDPs with Streett objectives. We first present previous results and then our contributions.

Previous results. The most basic algorithm for the problem for graphs is based on repeated SCC (strongly connected component) computation, and informally can be described as follows: for a given SCC, (a) if for every request type that is present in the SCC the corresponding grant type is also present in the SCC, then the SCC is identified as "good", (b) else vertices of each request type that have no corresponding grant type in the SCC are removed, and the algorithm recursively proceeds on the remaining graph. Finally, reachability to good SCCs is computed. The algorithm for MDPs is similar where the SCC computation is replaced with maximal end-component (MEC) computation, and reachability to good SCCs is replaced with probability 1 reachability to good MECs. The basic algorithms for graphs and MDPs with Streett objective have been improved in several works, such as for graphs in [19, 10], for MEC computation in [7, 8, 9], and MDPs with Streett objectives in [5]. For graphs/MDPs with n vertices, m edges, and k request-grant pairs with b denoting the size to describe the request grant pairs, the current best-known bound is $O(\min(n^2, m\sqrt{m \log n}) + b \log n)$.

Our contributions. In this work, our main contributions are randomized near-linear time (i.e. linear times a polylogarithmic factor) algorithms for graphs and MDPs with Streett objectives. In detail, our contributions are as follows:

- First, we present a near-linear time randomized algorithm for graphs with Streett objectives where the expected running time is $\widetilde{O}(m+b)$, where the $\widetilde{O}$ notation hides poly-log factors. Our algorithm is based on a recent randomized algorithm for maintaining the SCC decomposition of graphs under edge deletions, where the expected total running time is near linear [4].
- Second, by exploiting the results of [4] we present a randomized near-linear time algorithm for computing the MEC decomposition of an MDP where the expected running time is $\widetilde{O}(m)$. We extend the results of [4] from graphs to MDPs and present a randomized algorithm to maintain the MEC decomposition of an MDP under edge deletions, where the expected total running time is near linear [4].
- Finally, we use the result of the above item to present a near-linear time randomized algorithm for MDPs with Streett objectives where the expected running time is $\widetilde{O}(m+b)$.

All our algorithms are randomized and since they are near-linear in the size of the input, they are optimal up to poly-log factors. An important open question is whether there are deterministic algorithms that can improve the existing running time bound for graphs and MDPs with Streett objectives. Our algorithms are deterministic except for the invocation of the decremental SCC algorithm presented in [4].

Table 1 Summary of Results.

Problem	New Running Time	Old Running Time
Streett Objectives on Graphs	$\widetilde{O}(m+b)$	$\widetilde{O}(\min(n^2, m\sqrt{m})+b)$ [11, 19]
Almost-Sure Reachability	$\widetilde{O}(m)$	$O(m \cdot n^{2/3})$ [5, 9]
MEC Decomposition	$\widetilde{O}(m)$	$O(m \cdot n^{2/3})$ [9]
Decremental MEC Decomposition	$\widetilde{O}(m)$	$O(nm)$ [9]
Streett Objectives on MDPs	$\widetilde{O}(m+b)$	$\widetilde{O}(\min(n^2, m\sqrt{m})+b)$ [5]

2 Preliminaries

A *Markov decision process (MDP)* $P=((V,E),\langle V_1,V_R\rangle,\delta)$ consists of a finite set of vertices V partitioned into the player-1 vertices V_1 and the random vertices V_R, a finite set of edges $E \subseteq (V \times V)$, and a probabilistic transition function δ. The probabilistic transition function maps every random vertex in V_R to an element of $\mathcal{D}(V)$, where $\mathcal{D}(V)$ is the set of probability distributions over the set of vertices V. A random vertex v has an edge to a vertex $w \in V$, i.e. $(v,w) \in E$ iff $\delta(v)[w] > 0$. An edge $e=(u,v)$ is a *random edge* if $u \in V_R$ otherwise it is a *player-1 edge*. W.l.o.g. we assume $\delta(v)$ to be the uniform distribution over vertices u with $(v,u) \in E$.

Graphs are a special case of MDPs with $V_R = \emptyset$. The set $In(v)$ ($Out(v)$) describes the set of predecessors (successors) of a vertex v. More formally, $In(v)$ is defined as the set $\{w \in V \mid (w,v) \in E\}$ and $Out(v) = \{w \in V \mid (v,w) \in E\}$. When U is a set of vertices, we define $E(U)$ to be the set of all edges incident to the vertices in U. More formally, $E(U) = \{(u,v) \in E \mid u \in U \vee v \in U\}$. With $G[S]$ we denote the subgraph of a graph $G=(V,E)$ induced by the set of vertices $S \subseteq V$. Let $\mathsf{GraphReach}(S)$ be the set of vertices in G that can reach a vertex of $S \subseteq V$. The set $\mathsf{GraphReach}(S)$ can be found in linear time using depth-first search [27]. When a vertex u can reach another vertex v and vice versa, we say that u and v are *strongly connected*.

A *play* is an infinite sequence $\omega = \langle v_0, v_1, v_2, \ldots \rangle$ of vertices such that each $(v_{i-1}, v_i) \in E$ for all $i \geq 1$. The set of all plays is denoted with Ω. A play is initialized by placing a token on an initial vertex. If the token is on a vertex owned by player-1, he moves the token along one of the outgoing edges, whereas if the token is at a random vertex $v \in V_R$, the next vertex is chosen according to the probability distribution $\delta(v)$. The infinite sequence of vertices (infinite walk) formed in this way is a play.

Strategies are recipes for player 1 to extend finite prefixes of plays. Formally, a player-1 *strategy* is a function $\sigma : V^* \cdot V_1 \mapsto V$ which maps every finite prefix $\omega \in V^* \cdot V_1$ of a play that ends in a player-1 vertex v to a successor vertex $\sigma(\omega) \in V$, i.e., $(v, \sigma(\omega)) \in E$. A player-1 strategy is *memoryless* if $\sigma_1(\omega) = \sigma_1(\omega')$ for all $\omega, \omega' \in V^* \cdot V_1$ that end in the same vertex $v \in V_i$, i.e., the strategy does not depend on the entire prefix, but only on the last vertex. We write Σ for the set of all strategies for player 1.

The *outcome of strategies* is defined as follows: In graphs, given a starting vertex, a strategy for player 1 induces a unique play in the graph. In MDPs, given a starting vertex v and a strategy $\sigma \in \Sigma$, the outcome of the game is a random walk w_v^σ for which the probability of every event is uniquely defined, where an *event* $\mathcal{A} \subseteq \Omega$ is a measurable set of plays [28]. For a vertex v, strategy $\sigma \in \Sigma$ and an event $\mathcal{A} \subseteq \Omega$, we denote by $\Pr_v^\sigma(\mathcal{A})$ the probability that a play belongs to $\mathcal{A}$ if the game starts at v and player 1 follows σ.

An *objective* $\phi \subseteq \Omega$ for player 1 is an event, i.e., objectives describe the set of winning plays. A play $\omega \in \Omega$ *satisfies* the objective if $\omega \in \phi$. In MDPs, a player-1 strategy $\sigma \in \Sigma$ is almost-sure (a.s.) winning from a starting vertex $v \in V$ for an objective ϕ iff $\Pr_v^\sigma(\phi) = 1$. The *winning set* $\langle\langle 1 \rangle\rangle_{a.s.}(\phi)$ for player 1 is the set of vertices from which player 1 has an almost-sure winning strategy. We consider Reachability objectives and k-pair Streett objectives.

Given a set $T \subseteq V$ of vertices, the *reachability objective* $\mathit{Reach}(T)$ requires that some vertex in T be visited. Formally, the sets of winning plays are $\mathit{Reach}(T) = \{\langle v_0, v_1, v_2, \ldots \rangle \in \Omega \mid \exists k \geq 0 \text{ s.t. } v_k \in T\}$. We say v can reach u almost-surely (a.s.) if $v \in \langle\langle 1 \rangle\rangle_{a.s.}(\mathit{Reach}(\{u\}))$.

The *k-pair Streett objective* consists of k-Streett pairs
$(L_1, U_1), (L_2 U_2), \ldots, (L_k, U_k)$ where all $L_i, U_i \subseteq V$ for $1 \leq i \leq k$. An infinite path satisfies the objective iff for all $1 \leq i \leq k$ some vertex of L_i is visited infinitely often, then some vertex of U_i is visited infinitely often.

Given an MDP $P = (V, E, \langle V_1, V_R \rangle, \delta)$, an *end-component* is a set of vertices $X \subseteq V$ s.t. (1) the subgraph induced by X is strongly connected (i.e., $(X, E \cap X \times X)$ is strongly connected) and (2) all random vertices have their outgoing edges in X. More formally, for all $v \in X \cap V_R$ and all $(v, u) \in E$ we have $u \in X$. In a graph, if (1) holds for a set of vertices $X \subseteq V$ we call the set X strongly connected subgraph (SCS). An end-component, SCS respectively, is *trivial* if it only contains a single vertex with no edges. All other end-components, SCSs respectively, are *non-trivial.* A *maximal end-component* (MEC) is an end-component which is maximal under set inclusion. The importance of MECs is as follows: (i) it generalizes *strongly connected components* (SCCs) in graphs (with $V_R = \emptyset$) and closed recurrent sets of Markov chains (with $V_1 = \emptyset$); and (ii) in a MEC X, player-1 can almost-surely reach all vertices $u \in X$ from every vertex $v \in X$. The MEC-decomposition of an MDP is the partition of the vertex set into MECs and the set of vertices which do not belong to any MEC. The condensation of a graph G denoted by $\mathsf{CONDENSE}(G)$ is the graph where all vertices in the same SCC in G are contracted. The vertices of $\mathsf{CONDENSE}(G)$ are called *nodes* to distinguish them from the vertices in G.

Let C be a strongly connected component (SCC) of $G = (V, E)$. The SCC C is a *bottom SCC* if no vertex $v \in C$ has an edge to a vertex in $V \setminus C$, i.e., no outgoing edges. Consider an MDP $P = (V, E, \langle V_1, V_R \rangle, \delta)$ and notice that every bottom SCC C in the graph $G = (V, E)$ is a MEC because no vertex (and thus no random vertex) has an outgoing edge.

A *decremental graph algorithm* allows the deletion of player-1 edges while maintaining the solution to a graph problem. It usually allows three kinds of operations: (1) *preprocessing*, which is computed when the initial input is first received, (2) *delete*, which deletes a player-1 edge and updates the data structure, and (3) *query*, which computes the answer to the problem. The *query time* is the time needed to compute the answer to the query. The *update time* of a decremental algorithm is the cost for a *delete* operation. We sometimes refer to the delete operations as *update operation*. The running time of a decremental algorithm is characterized by the *total update time*, i.e., the sum of the update times over the entire sequence of deletions. Sometimes a decremental algorithm is randomized and assumes an *oblivious adversary* who fixes the sequence of updates in advance. When we use a decremental algorithm which assumes such an oblivious adversary as a subprocedure the sequence of deleted edges must not depend on the random choices of the decremental algorithm.

3 Decremental SCCs

We first recall the result about decremental strongly connected components maintenance in [4] (cf. Theorem 1 below) and then augment the result for our purposes.

▶ **Theorem 1** (Theorem 1.1 in [4]). *Given a graph $G = (V, E)$ with m edges and n vertices, we can maintain a data structure $\mathcal{A}$ that supports the operations*

- delete(u, v)*: Deletes the edge (u, v) from the graph G.*
- query(u, v)*: Returns whether u and v are in the same SCC in G,*

in total expected update time $O(m \log^4 n)$ and with worst-case constant query time. The bound holds against an oblivious adversary.

The preprocessing time of the algorithm is $O(m + n)$ using [27]. To use this algorithm we extend the query and update operations with three new operations described in Corollary 2.

▶ **Corollary 2.** *Given a graph $G = (V, E)$ with m edges and n vertices, we can maintain a data structure $\mathcal{A}$ that supports the operations*

- rep(u) *(query-operation): Returns a reference to the SCC containing the vertex u.*
- deleteAnnounce(E) *(update-operation): Deletes the set E of edges from the graph G. If the edge deletion creates new SCCs $C_1, \dots, C_k$ the operation returns a list $Q = \{C_1, \dots, C_k\}$ of references to the new SCCs.*
- deleteAnnounceNoOutgoing(E) *(update-operation): Deletes the set E of edges from the graph G. The operation returns a list $Q = \{C_1, \dots, C_k\}$ of references to all new SCCs with no outgoing edges.*

in total expected update time $O(m \log^4 n)$ and worst-case constant query time for the first operation. The bound holds against an oblivious adaptive adversary.

The first function is available in the algorithm described in [4]. The second function can be implemented directly from the construction of the data structure maintained in [4]. The key idea for the third function is that when an SCC splits, we consider the new SCCs. We distinguish between the largest of them and the others which we call small SCCs. We then consider all edges incident to the small SCCs: Note that as the new outgoing edges in the large SCC are also incident to a small SCC we can also determine the outgoing edges of the large SCC. Observe that whenever an SCC splits all the small SCCs are at most half the size of the original SCC. That is, each vertex can appear only $O(\log n)$ times in small SCCs during the whole algorithm. As an edge is only considered if one of the incident vertices is in a small SCC each edge is considered $O(\log n)$ times and the additional running time is bounded by $O(m \log n)$. Furthermore, we define T_d as the running time of the best decremental SCC algorithm which supports the operations in Corollary 2. Currently, $T_d = O(m \log^4 n)$.

4 Graphs with Streett Objectives

In this section, we present an algorithm which computes the winning regions for graphs with Streett objectives. The input is a directed graph $G = (V, E)$ and k Streett pairs (L_j, U_j) for $j = 1, \ldots, k$. The size of the input is measured in terms of $m = |E|$, $n = |V|$, k and $b = \sum_{j=1}^{k}(|L_j| + |U_j|) \leq 2nk$.

Algorithm Streett and good component detection. Let C be an SCC of G. In the good component detection problem, we compute (a) a non-trivial SCS $G[X] \subseteq C$ induced by the set of vertices X, such that for all $1 \leq j \leq k$ either $L_j \cap X = \emptyset$ or $U_j \cap X \neq \emptyset$ or (b) that no such SCS exists. In the first case, there exists an infinite path that eventually stays in X and satisfies the Streett objective, while in the latter case, there exists no path which satisfies the Streett objective in C. From the results of [1, Chapter 9, Proposition 9.4] the following algorithm, called Algorithm Streett, suffices for the winning set computation:

1. Compute the SCC decomposition of the graph;
2. For each SCC C for which the good component detection returns an SCS, label the SCC C as satisfying.
3. Output the set of vertices that can reach a satisfying SCC as the winning set.

Since the first and last step are computable in linear time, the running time of Algorithm Streett is dominated by the detection of good components in SCCs. In the following, we assume that the input graph is strongly connected and focus on the good component detection.

Bad vertices. A vertex is *bad* if there is some $1 \leq j \leq k$ such that the vertex is in L_j but it is not strongly connected to any vertex of U_j. All other vertices are good. Note that a good vertex might become bad if a vertex deletion disconnects an SCS or a vertex of a set U_j. A good component is a non-trivial SCS that contains only good vertices.

Decremental strongly connected components. Throughout the algorithm, we use the algorithm described in Section 3 to maintain the SCCs of a graph when deleting edges. In particular, we use Corollary 2 to obtain a list of the new SCCs which are created by removing bad vertices. Note that we can "remove" a vertex by deleting all its incident edges. Because the decremental SCC algorithm assumes an oblivious adversary we sort the list of the new SCCs as otherwise the edge deletions performed by our algorithm would depend on the random choices of the decremental SCC algorithm.

Data structure. During the course of the algorithm, we maintain a decomposition of the vertices in $G = (V, E)$: We maintain a list Q of certain sets $S \subseteq V$ such that every SCC of G is contained in some S stored in Q. The list Q provides two operations: $Q.\texttt{add}(X)$ enqueues X to Q; and $Q.\texttt{deque}()$ dequeues an arbitrary element X from Q. For each set S in the decomposition, we store a data structure $D(S)$ in the list Q. This data structure $D(S)$ supports the following operations

1. $\texttt{Construct}(S)$: initializes the data structure for the set S
2. $\texttt{Remove}(S, B)$ updates S to $S \setminus B$ for a set $B \subseteq V$ and returns $D(S)$ for the new set S.
3. $\texttt{Bad}(S)$ returns a reference to the set $\{v \in S \mid \exists j \text{ with } v \in L_j \text{ and } U_j \cap S = \emptyset\}$
4. $\texttt{SCCs}(S)$ returns the set of SCCs currently in $G[S]$. We implement $\texttt{SCCs}(S)$ as a balanced binary search tree which allows logarithmic and updates and deletions.

In [19] an implementation of this data structure with functions (1)-(3) is described that achieves the following running times. For a set of vertices $S \subseteq V$, let $\mathit{bits}(S)$ be defined as $\sum_{j=1}^{k}(|S \cap L_j| + |S \cap U_j|)$.

▶ **Lemma 3** (Lemma 2.1 in [19]). *After a one-time preprocessing of time $O(k)$, the data structure $D(S)$ can be implemented in time $O(\mathit{bits}(S) + |S|)$ for* `Construct`(S)*, time $O(\mathit{bits}(B) + |B|)$ for* `Remove`(S, B) *and constant running time for* `Bad`(S)*.*

We augment the data structure with the function `SCCs`(S) which runs in total time of a decremental SCC algorithm supporting the first function in Corollary 2.

Algorithm Description. The key idea is that the algorithm maintains the list Q of data structures $D(S)$ as described above when deleting bad vertices. Initially, we enqueue the data structure returned by `Construct`(V) to Q. As long as Q is non-empty, the algorithm repeatedly pulls a set S from Q and identifies and deletes bad vertices from $G[S]$. If no edge is contained in $G[S]$, the set S is removed as it can only induce trivial SCCs. Otherwise, the subgraph $G[S]$ is either determined to be strongly connected and output as a good component or we identify and remove an SCC with at most half of the vertices in $G[S]$. Consider Figure 1 for an illustration of an example run of Algorithm 1.

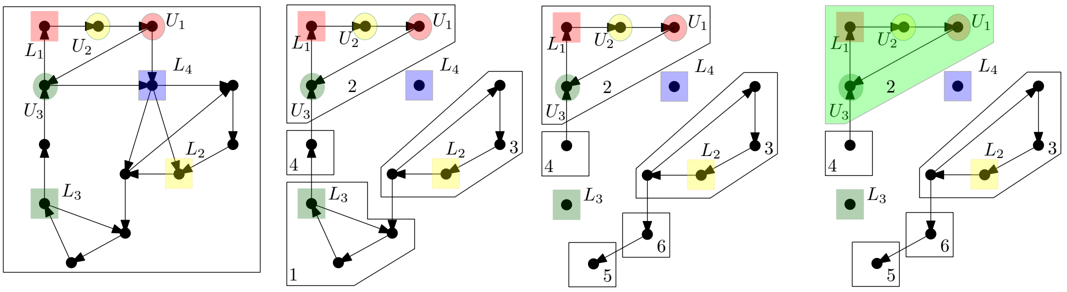


Figure 1 Illustration of one run of Algorithm 1: The vertex in the set L_4 is a bad vertex and we remove it from the SCC yielding four new SCCs. First, we look in the SCC containing L_3. The vertex in L_3 is a bad vertex because there is no vertex in U_3 in this SCC. Again two SCCs are created after its removal. The next SCC we process is the SCC containing L_1. It is a good component because the vertex in L_1 has a vertex in U_1 in the same SCC. No bad vertices are removed and the whole SCC is identified as a good component.

Outline correctness and running time. In the following, when we talk about the *input graph* $\hat{G}$ we mean the unmodified, strongly connected graph which we use to initialize Algorithm 1. In contrast, with the *current* graph G we refer to the graph where we already deleted vertices and their incident edges in the course of finding a good component. For the correctness of Algorithm 1, we show that if a good component exists, then there is a set S stored in list Q which contains all vertices of this good component.

To obtain the running time bound of Algorithm 1, we use the fact that we can maintain the SCC decomposition under deletions in $O(T_d)$ total time. With the properties of the data structure described in Lemma 3 we get a running time of $\widetilde{O}(n + b)$ for the maintenance of the data structure and identification of bad vertices over the whole algorithm. Combined, these ideas lead to a total running time of $\widetilde{O}(T_d + n + b)$ which is $\widetilde{O}(m + b)$ using Corollary 2.

▶ **Lemma 4.** *Algorithm 1 runs in expected time $\widetilde{O}(m + b)$.*

Algorithm 1: Algorithm GoodComp.

```
   Input: Strongly connected graph $G = (V, E)$ and Streett pairs $(L_j, U_j)$ for $j = 1, \dots, k$
   Output: a good component in $G$ if one exists
1  Invoke an instance $\mathcal{A}$ of the decremental SCC algorithm; Initialize $Q$ as a new list.;
2  $D(V) = \texttt{Construct}(V)$; $D(V).\texttt{SCCs}(V) \leftarrow \{\mathcal{A}.\mathsf{rep}(x)\}$ for some $x \in V$;
3  $Q.\texttt{add}(D(V))$;
4  while $Q$ is not empty do
5      $D(S) \leftarrow Q.\texttt{deque}()$;
6      while $D(S).\texttt{Bad}(S)$ is not empty do
7          $B \leftarrow D(S).\texttt{Bad}(S)$; $D(S) \leftarrow D(S).\texttt{Remove}(S, B)$;
           // obtain SCCs after deleting bad vertices from $S$
8          $D(S).\texttt{SCCs}(S) \leftarrow D(S).\texttt{SCCs}(S) \setminus \left(\bigcup_{b \in B}\{\mathcal{A}.\mathsf{rep}(b)\}\right)$;
9          $D(S).\texttt{SCCs}(S) \leftarrow D(S).\texttt{SCCs}(S) \cup \mathcal{A}.\mathsf{deleteAnnounce}(E(B))$;
10     if $G[S]$ contains at least one edge then
11         Initialize $K$ as a new list;
12         for $X \leftarrow D(S).\texttt{SCCs}(S)$ do
13             if $X = S$ then output $G[S]$;                    // good component found
14             if $|X| \le \frac{|S|}{2}$ then $K.\texttt{add}(X)$;
15         Sort the SCCs in $K$ by vertex id (look at all the vertices in each SCC of $K$);
16         $R \leftarrow \emptyset$;        // Build $D(X)$ for SCCs $X$ in $K$ and remove $X$ from $S$,$D(S)$ and
            SCCs($S$)
17         for $X \leftarrow K.\texttt{deque}()$ do
18             $R \leftarrow R \cup X$; $D(X) = \texttt{Construct}(X)$;
19             $D(X).\texttt{SCCs}(X) \leftarrow \{\mathcal{A}.\mathsf{rep}(x)\}$ for some $x \in X$;
20             $D(S).\texttt{SCCs}(S) \leftarrow D(S).\texttt{SCCs}(S) \setminus \{\mathcal{A}.\mathsf{rep}(x)\}$ for some $x \in X$;
21             $Q.\texttt{add}(D(X))$;
22         if $D(S).\texttt{SCCs}(S) \neq \emptyset$ then $Q.\texttt{add}(D(S).\texttt{Remove}(S, R))$ ;
23 return No good component exists.
```

Proof. The preprocessing and initialization of the data structure D and the removal of bad vertices in the whole algorithm takes time $O(m + k + b)$ using Lemma 3. Since each vertex is deleted at most once, the data structure can be constructed and maintained in total time $O(m)$. Announcing the new SCCs after deleting the bad vertices at Line 9 is in $O(T_d) = \widetilde{O}(m)$ total time by Corollary 2. Consider an iteration of the while loop at Line 4: A set S is removed from Q. Let us denote by n' the number of vertices of S. If $G[S]$ does not contain any edge after the removal of bad vertices, then S is not considered further by the algorithm. Otherwise, the for-loop at Line 12 considers all new SCCs. Note the we can implement the for-loop in a lockstep fashion: In each step for each SCC we access the i-th vertex and as soon as all of the vertices of an SCC are accessed we add it to the list K. When only one SCC is left we compute its size using the original set S and the sizes of the other SCCs. If its size is at most $|S|/2$ we add it to K. Note that this can be done in time proportional to the number of vertices in the SCCs in S of size at most $|S|/2$. The sorting operation at Line 15 takes time $O(|K| \log |K|)$ plus the size of all the SCCs in K, that is $\sum_{K_i \in K} |K_i|$. Note that $O(|K| \log |K|) = O((\sum_{K_i \in K} |K_i|) \log(\sum_{K_i \in K} |K_i|))$. Let $K_i \in K$ be an SCC stored in K. Note that during the algorithm each vertex can appear at most $\log(n)$ times in the list K. This is by the fact that K only contains SCCs that are at most half the size of the original set S. We obtain a running time bound of $O(n(\log n)^2)$ for Lines 12-15.

Consider the second for-loop at Line 17: Let $|X| = n_1$. The operations $\texttt{Remove}(\cdot)$ and $\texttt{Construct}(\cdot)$ are called once per found SCC $G[X]$ with $X \neq S$ and take by Lemma 3 $O(|X| + \mathit{bits}(X))$ time. Whenever a vertex is in X, the size of the set in Q containing v originally is reduced by at least a factor of two due to the fact that $|X| = n_1 \leq n'/2$. This happens at most $\lceil \log n \rceil$ times. By charging $O(1)$ to the vertices in X and, respectively, to $\mathit{bits}(X)$, the total running time for Lines 21 & 22 can be bounded by $O((n + b) \log n)$ as each vertex and bit is only charged $O(\log n)$ times. Combining all parts yields the claimed running time bound of $O(T_d + b \log n + n \log^2 n) = \widetilde{O}(m + b)$. ◀

The correctness of the algorithm is similar to the analysis given in [11, Lemmas 3.6 & 3.7] except that we additionally have to prove that $\texttt{SCCs}(S)$ holds the SCCs of $G[S]$. Lemma 5 shows that we maintain $\texttt{SCCs}(S)$ properly for all the data structures in Q.

▶ **Lemma 5.** *After each iteration of the outer while-loop every non-trivial SCC of the current graph is contained in one of the subgraphs $G[S]$ for which the data structure $D(S)$ is maintained in Q and $\texttt{SCCs}(S)$ stores a list of all SCCs contained in S.*

We prove the next Lemma by showing that we never remove edges of vertices of good components.

▶ **Lemma 6.** *After each iteration of the outer while-loop every good component of the input graph is contained in one of the subgraphs $G[S]$ for which the data structure $D(S)$ is maintained in the list Q.*

▶ **Proposition 7.** *Algorithm 1 outputs a good component if one exists, otherwise the algorithm reports that no such component exists.*

Proof. First consider the case where Algorithm 1 outputs a subgraph $G[S]$. We show that $G[S]$ is a good component: Line 10 ensures only non-trivial SCSs are considered. After the removal of bad vertices from S in Lines 6-9, we know that for all $1 \leq j \leq k$ that $U_j \cap S \neq \emptyset$ if $S \cap L_j \neq \emptyset$. Due to Line 13 there is only one SCC in $G[S]$ and thus $G[S]$ is a good component. Second, if Algorithm 1 terminates without a good component, by Lemma 6, we have that the initial graph has no good component and thus the result is correct as well. ◀

The running time bounds for the decremental SCC algorithm of [4] (cf. Corollary 2) only hold against an oblivious adversary. Thus we have to show that in our algorithm the sequence of edge deletions does not depend on the random choices of the decremental SCC algorithm. The key observation is that only the order of the computed SCCs depends on the random choices of the decremental SCC and we eliminate this effect by sorting the SCCs.

▶ **Proposition 8.** *The sequence of deleted edges does not depend on the random choices of the decremental SCC Algorithm but only on the given instance.*

Due to Lemma 4, Lemma 7 and Proposition 8 we obtain the following result.

▶ **Theorem 9.** *In a graph, the winning set for a k-pair Streett objective can be computed in $\widetilde{O}(m + b)$ expected time.*

5 Algorithms for MDPs

In this section, we present expected near-linear time algorithms for computing a MEC decomposition, deciding almost-sure reachability and maintaining a MEC decomposition in a decremental setting. In the last section, we present an algorithm for *MDPs* with Streett objectives by using the new algorithm for the decremental MEC decomposition.

Random attractor. First, we introduce the notion of a *random attractor* $\mathsf{attr_R(T)}$ for a set $T \subseteq V$. The random attractor $A = \mathsf{attr_R(T)}$ is defined inductively as follows: $A_0 = T$ and $A_{i+1} = A_i \cup \{v \in V_R \mid Out(v) \cap A_i \neq \emptyset\} \cup \{v \in V_1 \mid Out(v) \subseteq A_i\}$ for all $i > 0$. Given a set T, the random attractor includes all vertices (1) in T, (2) random vertices with an edge to A_i, (3) player-1 vertices with all outgoing edges in A_i. Due to [21, 3] we can compute the random attractor $A = \mathsf{attr_R(S)}$ of a set S in time $O(\sum_{v \in A} In(v))$.

5.1 Maximal End-Component Decomposition

In this section, we present an expected near linear time algorithm for MEC decomposition. Our algorithm is an efficient implementation of the static algorithm presented in [9, p. 29]: The difference is that the bottom SCCs are computed with a dynamic SCC algorithm instead of recomputing the static SCC algorithm. A similar algorithm was independently proposed in an unpublished extended version of [12].

Algorithm Description. The MEC algorithm described in Algorithm 2 repeatedly removes bottom SCCs and the corresponding random attractor. After removing bottom SCCs the new SCC decomposition with its bottom SCCs is computed using a dynamic SCC algorithm.

Algorithm 2: MEC Algorithm.

```
  Input: MDP $P = (V, E, \langle V_1, V_R \rangle, \delta)$, decremental SCC algorithm $\mathcal{A}$
1 Invoke an instance $\mathcal{A}$ of the decremental SCC algorithm;
2 Compute the SCC-decomposition of $G = (V, E)$: $C = \{C_1, \ldots, C_\ell\}$ ;
3 Let $M = \emptyset$; $Q \leftarrow \{C_i \in C \mid C_i$ has no outgoing edges$\}$;
4 while $Q$ is not empty do
5     $C \leftarrow \emptyset$;
6     for $C_k \in Q$ do $C \leftarrow C \cup C_k$; $M \leftarrow M \cup \{C_k\}$ ;
7     $A \leftarrow \mathsf{attr_R(C)}$;
8     $Q \leftarrow \mathcal{A}.\mathsf{deleteAnnounceNoOutgoing}(E(A))$;
9 return $M$;
```

Correctness follows because our algorithm just removes attractors of bottom SCCs and marks bottom SCCs as MECs. This is precisely the second static algorithm presented in [9, p. 29] except that the bottom SCCs are computed using a dynamic data structure. By using the decremental SCC algorithm described in Subsection 3 we obtain the following lemma.

▶ **Lemma 10.** *Algorithm 2 returns the MEC-decomposition of an MDP P in expected time $\widetilde{O}(m)$.*

Proof. The running time of algorithm $\mathcal{A}$ is in total time $O(T_d) = \widetilde{O}(m)$ by Theorem 1 and Corollary 2. Initially, computing the SCC decomposition and determining the SCCs with no outgoing edges takes time $O(m + n)$ by using [27]. Each time we compute the attractor of a bottom SCC C_k at Line 7 we remove it from the graph by deleting all its edges and never process these edges and vertices again. Since we can compute the attractor A at Line 7 in time $O(\sum_{v \in A} In(A))$, we need $O(m + n)$ total time for computing the attractors of all bottom SCCs. Hence, the running time is dominated by the decremental SCC algorithm $\mathcal{A}$, which is $O(T_d) = \widetilde{O}(m)$. ◀

The algorithm uses $O(m + n)$ space due to the fact that the decremental SCC algorithm $\mathcal{A}$ uses $O(m + n)$ space and Q only contains vertices.

▶ **Theorem 11.** *Given an MDP the MEC-decomposition can be computed in $\widetilde{O}(m)$ expected time. The algorithm uses $O(m+n)$ space.*

Note that we can use the decremental SCC Algorithm $\mathcal{A}$ of [4] even though this algorithm only works against an oblivious adversary as the sequence of deleted edges does not depend on the random choices of the decremental SCC Algorithm.

5.2 Almost-Sure Reachability

In this section, we present an expected near linear-time algorithm for the almost-sure reachability problem. In the almost-sure reachability problem, we are given an MDP P and a target set T and we ask for which vertices player 1 has a strategy to reach T almost surely, i.e., $\langle\langle 1\rangle\rangle_{a.s.}(\mathit{Reach}(T))$. Due to [5, Theorem 4.1] we can determine the set $\langle\langle 1\rangle\rangle_{a.s.}(\mathit{Reach}(T))$ in time $O(m + \mathsf{MEC})$ where MEC is the running time of the fastest MEC algorithm. We use Theorem 11 to compute the MEC decomposition and obtain the following theorem.

▶ **Theorem 12.** *Given an MDP and a set of vertices T we can compute $\langle\langle 1\rangle\rangle_{a.s.}(\mathit{Reach}(T))$ in $\widetilde{O}(m)$ expected time.*

5.3 Decremental Maximal End-Component Decomposition

We present an expected near-linear time algorithm for the MEC-decomposition which supports player-1 edge deletions and a query that answers if two vertices are in the same MEC. We need the following lemma from [7] to prove the correctness of our algorithm. Given an SCC C we consider the set U of the random vertices in C with edges leaving C. The lemma states that for all non-trivial MECs X in P the intersection with U is empty, i.e., $\mathsf{attr}_\mathsf{R}(\mathsf{U}) \cap X = \emptyset$.

▶ **Lemma 13** (Lemma 2.1(1), [7])**.** *Let C be an SCC in P. Let $U = \{v \in C \cap V_R \mid E(v) \cap (V \setminus C) \neq \emptyset\}$ be the random vertices in C with edges leaving C. Let $Z = \mathsf{attr}_\mathsf{R}(\mathsf{U}) \cap C$. Then for all non-trivial MECs X in P we have $Z \cap X = \emptyset$ and for any edge (u, v) with $u \in X$ and $v \in Z$, u must belong to V_1.*

The *pure MDP graph* P^P of an MDP $P = (V, E, \langle V_1, V_R\rangle, \delta)$ is the graph which contains only edges in non-trivial MECs of P. More formally, the pure MDP graph P^P is defined as follows: Let $M_1, \dots M_k$ be the set of MECs of P. $P^P = (V^P, E^P, \langle V_1^P, V_R^P\rangle, \delta^P)$ where $V^P = V, V_1^P = V_1, V_R^P = V_R$, $E^P = \bigcup_{i=1}^k \{(u, v) \in E \cap (M_i \times M_i)\}$ and for each $v \in V_R$: $\delta^P(v)$ the uniform distribution over vertices u with $(v, u) \in E^P$.

Throughout the algorithm, we maintain the pure MDP graph P^P for an input MDP P. Note that every non-trivial SCC in P^P is also a MEC due to the fact that there are only edges inside of MECs. Moreover, a trivial SCC $\{v\}$ is a MEC iff $v \in V_1$. Note furthermore that when a player-1 edge of an MDP P is deleted, existing MECs might split up into several MECs but no new vertices are added to existing MECs.

Initially, we compute the MEC-decomposition in $\widetilde{O}(m)$ expected time using the algorithm described in Section 5.1. Then we remove every edge that is not in a MEC. The resulting graph is the pure MDP graph P^P. Additionally, we invoke a decremental SCC algorithm $\mathcal{A}$ which is able to (1) announce new SCCs under edge deletions and return a list of their vertices and (2) is able to answer queries that ask whether two vertices v, u belong to the same SCC. When an edge (u, v) is deleted, we know that (i) the MEC-decomposition stays the same or (ii) one MEC splits up into new MECs and the rest of the decomposition stays the same. We first check if u and v are in the same MEC, i.e., if it exists in P^P. If not, we are done. Otherwise, u and v are in the same MEC C and either (1) the MEC C does

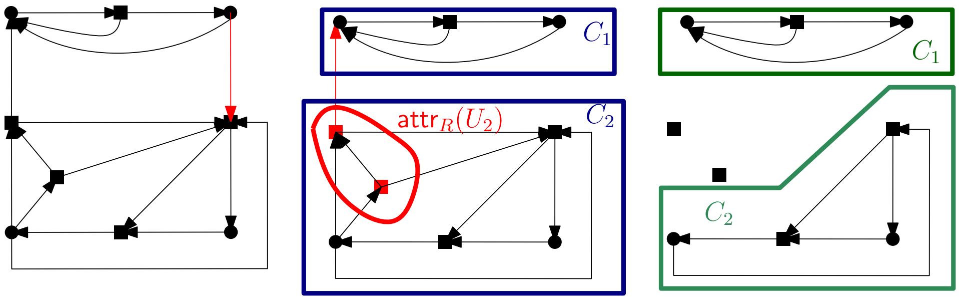


Figure 2 We delete an edge which splits the MEC into two new SCCs C_1 and C_2. The SCC C_2 is not a MEC. We thus compute and remove the attractor of U_2 and the resulting SCC is a MEC.

not split or (2) the MEC C splits. In the case of (1) the SCCs of the pure MDP graph P^P remain intact and nothing needs to be done. In the case of (2) we need to identify the new SCCs $C_1, \dots, C_k$ in P^P using the decremental SCC algorithm $\mathcal{A}$. Let, w.l.o.g., C_1 be the SCC with the most vertices. We iterate through every edge of the vertices in the SCCs $C_2, \dots, C_k$. By considering all the edges, we identify all SCCs (including C_1) which are also MECs. We remove all edges (y, z) where y and z are not in the same SCC to maintain the pure MDP graph P^P. For the SCCs that are not MECs let U be the set of random vertices with edges leaving its SCC. We compute and remove $A = \mathsf{attr}_\mathsf{R}(\mathsf{U})$ (these vertices belong to no MEC due to Lemma 13) and recursively start the procedure on the new SCCs generated by the deletion of the attractor. The algorithm is illustrated in Figure 2.
Lemma 14 describes the key invariants of the while-loop at Line 3. We prove it with a straightforward induction on the number of iterations of the while-loop and apply Lemma 13.

▶ **Lemma 14.** *Assume that $\mathcal{A}$ maintains the pure MDP graph P^P before the deletion of $e = (u, v)$ then the while-loop at Line 3 maintains the following invariants:*

1. *For the graph stored in $\mathcal{A}$ and all lists of SCCs $\{C_1, \dots, C_k\}$ in K there are only edges inside the SCCs or between the SCCs in the list, i.e., for each $(x, y) \in \bigcup_{j=0}^{k} E[C_j]$ we have $x, y \in \bigcup_{j=0}^{k} C_j$.*
2. *If a non-trivial SCC of the graph in $\mathcal{A}$ is not a MEC of the current MDP it is in K.*
3. *If M is a MEC of the current MDP then we do not delete an edge of M in the while-loop.*

▶ **Proposition 15.** *Algorithm 3 maintains the pure MDP graph P^P in the data structure $\mathcal{A}$ under player-1 edge deletions.*

Proof. We show that after deleting an edge using Algorithm 3 (i) every non-trivial SCC is a MEC and vice-versa, and (ii) there are no edges going from one MEC to another. Initially, we compute the pure MDP graph and both conditions are fulfilled.

When we delete an edge and the while-loop at Line 3 terminates (i) is true due to Lemma 14(2,3). That is, as we never delete edges within MECs they are still strongly connected and when the while-loop terminates, $K = \emptyset$ which means that all SCCs are MECs.

For (ii) notice that each SCC is once processed as a List J. Consider an arbitrary SCC C_i and the corresponding list of SCCs $J = \{C_1, \dots, C_k\}$ of the iteration in which C_i was identified as a MEC. By Lemma 14(1) there are no edges to SCCs not in the list. Additionally, due to Line 10 we remove all edges from C_i to other SCCs in J. ◀

Now that we maintain the pure MDP graph P^P in $\mathcal{A}$, we can answer MEC queries of the form: $\mathsf{query}(u, v)$: Returns whether u and v are in the same MEC in P, by an SCC query $\mathcal{A}.\mathsf{query}(u, v)$ on the pure MDP graph P^P.

Algorithm 3: Decremental MEC-update.

```
Input: Player-1 Edge $e = (u, v)$
if $\mathcal{A}$.query$(u, v) = true$ then
    List $K \leftarrow \{\mathcal{A}$.deleteAnnounce$((u, v))\}$;
    while $K \neq \emptyset$ do
        pull a list $J$ of SCCs from $K$ and let $C_1$ be the largest SCC;
        $\{C_1, \dots C_k\} \leftarrow$ Sort all SCCs in $J$ except $C_1$ by the smallest vertex id.;
        $\mathsf{MEC}^{C_1} =$ True, $U_1 \leftarrow \emptyset$;
        for $i = 2;\ i \leq k;\ i{+}{+}$ do
            $\mathsf{MEC}^{C_i} =$ True, $U_i \leftarrow \emptyset$;
            for $e = (s, t)$ where $e \in E(C_i)$ do
                if $(s \notin C_i) \vee (t \notin C_i)$ then $\mathcal{A}$.delete$(e)$ ;
                if $(s \in V_R \wedge t \notin C_i)$ then $\mathsf{MEC}^{C_i} =$ False; $U_i \leftarrow U_i \cup \{s\}$ ;
                if $(s \in V_R \wedge s \in C_1)$ then $\mathsf{MEC}^{C_1} =$ False; $U_1 \leftarrow U_1 \cup \{s\}$ ;
            if $\mathsf{MEC}^{C_i} =$ False then
                $A \leftarrow \mathsf{attr_R(U_i)} \cap C_i$;
                $J \leftarrow \mathcal{A}$.deleteAnnounce$(E(A)) \setminus \left(\bigcup_{a \in A} \mathcal{A}.\mathsf{rep}(a)\right)$;
                if $J \neq \emptyset$ then $K \leftarrow K \cup \{J\}$ ;
        if $\mathsf{MEC}^{C_1} =$ False then
            $A \leftarrow \mathsf{attr_R(U_1)} \cap C_1$;
            $J \leftarrow \mathcal{A}$.deleteAnnounce$(E(A)) \setminus \left(\bigcup_{a \in A} \mathcal{A}.\mathsf{rep}(a)\right)$;
            if $J \neq \emptyset$ then $K \leftarrow K \cup \{J\}$ ;
```

The key idea for the running time of Algorithm 3 is that we do not look at edges of the largest SCCs but the new SCC decomposition by inspecting the edges of the smaller SCCs. Note that we identify the largest SCC by processing the SCCs in a lockstep manner. This can only happen $\lceil \log n \rceil$ times for each edge. Additionally, when we sort the SCCs, we only look at the vertex ids of the smaller SCCs and when we charge this cost to the vertices we need $O(n \log^2 n)$ additional time.

▶ **Proposition 16.** *Algorithm 3 maintains the MEC-decomposition of P under player-1 edge deletions in expected total time $\widetilde{O}(m)$. Algorithm 3 answers queries that ask whether two vertices v, u belong to the same MEC in $O(1)$. The algorithm uses $O(m + n)$ space.*

Due to the fact that the decremental SCC algorithm we use in Corollary 2 only works for an oblivious adversary, we prove the following proposition. The key idea is that we sort SCCs returned by the decremental SCC Algorithm. Thus, the order in which new SCCs are returned does only depend on the given instance.

▶ **Proposition 17.** *The sequence of deleted edges does not depend on the random choices of the decremental SCC Algorithm but only on the given instance.*

The algorithm presented in [4] fulfills all the conditions of Proposition 16 due to Corollary 2. Therefore we obtain the following theorem due to Proposition 15 and Proposition 16.

▶ **Theorem 18.** *Given an MDP with n vertices and m edges, the MEC-decomposition can be maintained under the deletion of $O(m)$ player-1 edges in total expected time $\widetilde{O}(m)$ and we can answer queries that ask whether two vertices v, u belong to the same MEC in $O(1)$ time. The algorithm uses $O(m + n)$ space. The bound holds against an oblivious adversary.*

5.4 MDPs with Streett Objectives

Similar to graphs we compute the winning region of Streett objectives with k pairs (L_i, U_i) (for $1 \leq i \leq k$) for an MDP P as follows:

1. We compute the MEC-decomposition of P.
2. For each MEC, we find good end-components, i.e., end-components where $L_i \cap X = \emptyset$ or $U_i \cap X \neq \emptyset$ for all $1 \leq i \leq k$ and label the MEC as satisfying.
3. We output the set of vertices that can almost-surely reach a satisfying MECs.

For 2., we find good *end-components* similar to how we find good components as in Section 4. The key idea is to use the decremental MEC-Algorithm described in Section 5.3 instead of the decremental SCC Algorithm. We modify the Algorithm presented in Section 4 as follows to detect good end-components: First, we use the decremental MEC-algorithm instead of the decremental SCC Algorithm. Towards this goal, we augment the decremental MEC-algorithm with a function to return a list of references to the new MECs when we delete a set of edges. Second, the decremental MEC-algorithm does not allow the deletion of arbitrary edges, but only player-1 edges. To overcome this obstacle, we create an equivalent instance where we remove player-1 edges when we remove "bad" vertices.

▶ **Lemma 19.** *Given an MDP $P = (V, E, \langle V_1, V_R\rangle, \delta)$ with m edges and n vertices, we can maintain a data structure that supports the operation*

- deleteAnnounce(E)*: Deletes the set of E of player-1 edges (u, v) from the MDP P. If the edge deletion creates new MECs $C_1, \ldots, C_k$ the operation returns a list $Q = \{C_1, \ldots, C_k\}$ of references to the new non-trivial MECs.*

in total expected update time $\widetilde{O}(m)$. The bound holds against an oblivious adaptive adversary.

Deleting bad vertices. As the decremental MEC-algorithm only allows deletion of player-1 edges, we first modify the original instance $P = (V, E, \langle V_1, V_R\rangle, \delta)$ to a new instance $P' = (V', E', \langle V_1', V_R'\rangle, \delta')$ such that we can remove bad vertices by deleting player-1 edges only. In P' each vertex $v \in V_x$ for $x \in \{1, R\}$ is split into two vertices $v_{in} \in V_1'$ and $v_{out} \in V_x'$ such that $E' = \{(u_{out}, v_{in}) \mid (u, v) \in E\} \cup \{(v_{in}, v_{out}) \mid v \in V\}$ and $L_i' = \{v_{in} \in V' \mid v \in L_i\}$ and $U_i' = \{v_{out} \in V' \mid v \in U_i\}$ for all $1 \leq i \leq k$. The new probability distribution is $\delta'(v_{out})[w_{in}] = \delta(v)[w]$ for $v \in V_R$ and $w \in Out(v)$. Note that for each $v \in V_R$ the corresponding vertex $v_{out} \in V_R'$ has the same probabilities to reach the representation v_{out} of a vertex as v. The described reduction allows us to remove bad vertices from MECs by removing the player-1 edge (v_{in}, v_{out}).

The key idea for the following lemma is that for each original vertex $v \in V$ either both v_{in} and v_{out} are part of a good end-component or none of them. Note that the only way that v_{in} and v_{out} are strongly connected is when the other vertex is also in the strongly connected component because v_{in} (v_{out}) has only one outgoing (incoming) edges to v_{out} (from v_{in}).

▶ **Lemma 20.** *There is a good end-component in the modified instance P' iff there is a good component in the original instance P.*

On the modified instance P' the algorithm for MDPs is identical to Algorithm 1 except that we use a dynamic MEC algorithm instead of a dynamic SCC algorithm.

▶ **Theorem 21.** *In an MDP the winning set for a k-pair Street objectives can be computed in $\widetilde{O}(m + b)$ expected time.*

References

1 R. Alur and T.A. Henzinger. Computer-Aided Verification. unpublished., 2004. URL: `https://web.archive.org/web/20041207121830/http://www.cis.upenn.edu/group/cis673/`.

2 C. Baier and J.P. Katoen. *Principles of model checking*. MIT Press, 2008.

3 C. Beeri. On the Membership Problem for Functional and Multivalued Dependencies in Relational Databases. *ACM Trans. Database Syst.*, 5(3):241–259, 1980. `doi:10.1145/320613.320614`.

4 A. Bernstein, M. Probst, and C. Wulff-Nilsen. Decremental Strongly-Connected Components and Single-Source Reachability in Near-Linear Time. In *STOC*, pages 365–376, 2019. `doi:10.1145/3313276.3316335`.

5 K. Chatterjee, W. Dvořák, M. Henzinger, and V. Loitzenbauer. Model and Objective Separation with Conditional Lower Bounds: Disjunction is Harder than Conjunction. In *LICS*, pages 197–206, 2016. `doi:10.1145/2933575.2935304`.

6 K. Chatterjee, A. Gaiser, and J. Kretínský. Automata with Generalized Rabin Pairs for Probabilistic Model Checking and LTL Synthesis. In *CAV*, pages 559–575, 2013. `doi:10.1007/978-3-642-39799-8_37`.

7 K. Chatterjee and M. Henzinger. Faster and Dynamic Algorithms for Maximal End-Component Decomposition and Related Graph Problems in Probabilistic Verification. In *SODA*, pages 1318–1336, 2011. `doi:10.1137/1.9781611973082.101`.

8 K. Chatterjee and M. Henzinger. An $O(n^2)$ Time Algorithm for Alternating Büchi Games. In *SODA*, pages 1386–1399, 2012. URL: `http://portal.acm.org/citation.cfm?id=2095225&CFID=63838676&CFTOKEN=79617016`.

9 K. Chatterjee and M. 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`.

10 K. Chatterjee, M. Henzinger, and V. Loitzenbauer. Improved Algorithms for One-Pair and k-Pair Streett Objectives. In *LICS*, pages 269–280, 2015. `doi:10.1109/LICS.2015.34`.

11 K. Chatterjee, M. Henzinger, and V. Loitzenbauer. Improved Algorithms for Parity and Streett objectives. *Logical Methods in Computer Science*, 13(3):1–27, 2017. `doi:10.23638/LMCS-13(3:26)2017`.

12 S. Chechik, T. D. Hansen, G. F. Italiano, J. Lacki, and N. Parotsidis. Decremental Single-Source Reachability and Strongly Connected Components in $\tilde{\mathrm{O}}(\mathrm{m}\sqrt{\mathrm{n}})$ Total Update Time. In *FOCS*, pages 315–324, 2016. `doi:10.1109/FOCS.2016.42`.

13 F. Ciesinski and C. Baier. LiQuor: A Tool for Qualitative and Quantitative Linear Time Analysis of Reactive Systems. In *QEST*, pages 131–132, 2006. `doi:10.1109/QEST.2006.25`.

14 A. Cimatti, E. Clarke, F. Giunchiglia, and M. Roveri. NUSMV: A new Symbolic Model Checker. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(4):410–425, 2000. `doi:10.1007/s100090050046`.

15 E. M. Clarke, Jr., O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, Cambridge, MA, USA, 1999.

16 C. Courcoubetis and M. Yannakakis. The Complexity of Probabilistic Verification. *J. ACM*, 42(4):857–907, 1995. `doi:10.1145/210332.210339`.

17 C. Dehnert, S. Junges, J.P. Katoen, and M. Volk. A Storm is Coming: A Modern Probabilistic Model Checker. In *CAV*, pages 592–600, 2017. `doi:10.1007/978-3-319-63390-9_31`.

18 J. Esparza and J. Kretínský. From LTL to Deterministic Automata: A Safraless Compositional Approach. In *CAV*, pages 192–208, 2014. `doi:10.1007/978-3-319-08867-9_13`.

19 M. Rauch Henzinger and J. A. Telle. Faster Algorithms for the Nonemptiness of Streett Automata and for Communication Protocol Pruning. In *SWAT*, pages 16–27, 1996. `doi:10.1007/3-540-61422-2_117`.

20 G. J. Holzmann. The Model Checker SPIN. *IEEE Trans. Softw. Eng.*, 23(5):279–295, 1997. `doi:10.1109/32.588521`.

21 N. Immerman. Number of Quantifiers is Better Than Number of Tape Cells. *J. Comput. Syst. Sci.*, 22(3):384–406, 1981. doi:10.1016/0022-0000(81)90039-8.

22 Z. Komárková and J. Kretínský. Rabinizer 3: Safraless Translation of LTL to Small Deterministic Automata. In *ATVA*, pages 235–241, 2014. doi:10.1007/978-3-319-11936-6_17.

23 M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *CAV*, pages 585–591, 2011. doi:10.1007/978-3-642-22110-1_47.

24 Z. Manna and A. Pnueli. Temporal Verification of Reactive Systems: Progress (Draft), 1996. URL: http://theory.stanford.edu/~zm/tvors3.html.

25 S. Safra. On the Complexity of ω-Automata. In *FOCS*, pages 319–327, 1988. doi:10.1109/SFCS.1988.21948.

26 R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems*. PhD thesis, MIT, 1995.

27 R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

28 M. Y. Vardi. Automatic Verification of Probabilistic Concurrent Finite-State Programs. In *FOCS*, pages 327–338, 1985. doi:10.1109/SFCS.1985.12.

Life Is Random, Time Is Not: Markov Decision Processes with Window Objectives

Thomas Brihaye
UMONS – Université de Mons, Belgium

Florent Delgrange
UMONS – Université de Mons, Belgium
RWTH Aachen, Germany

Youssouf Oualhadj
LACL – UPEC, Paris, France

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

Abstract

The window mechanism was introduced by Chatterjee et al. [17] to strengthen classical game objectives with time bounds. It permits to synthesize system controllers that exhibit acceptable behaviors within a configurable time frame, all along their infinite execution, in contrast to the traditional objectives that only require correctness of behaviors in the limit. The window concept has proved its interest in a variety of two-player zero-sum games, thanks to the ability to reason about such time bounds in system specifications, but also the increased tractability that it usually yields. In this work, we extend the window framework to stochastic environments by considering the fundamental *threshold probability problem* in Markov decision processes for window objectives. That is, given such an objective, we want to synthesize strategies that guarantee satisfying runs with a given probability. We solve this problem for the usual variants of window objectives, where either the time frame is set as a parameter, or we ask if such a time frame exists. We develop a generic approach for window-based objectives and instantiate it for the classical mean-payoff and parity objectives, already considered in games. Our work paves the way to a wide use of the window mechanism in stochastic models.

2012 ACM Subject Classification Software and its engineering → Formal methods; Theory of computation → Logic and verification; Theory of computation → Markov decision processes

Keywords and phrases Markov decision processes, window mean-payoff, window parity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.8

Related Version Full version available at `https://arxiv.org/abs/1901.03571`.

Funding Research supported by F.R.S.-FNRS, Grant n° F.4520.18 (ManySynth), and F.R.S.-FNRS mobility funding for scientific missions (Y. Oualhadj in UMONS, 2018).
Mickael Randour: F.R.S.-FNRS Research Associate.

1 Introduction

Game-based models for controller synthesis. *Two-player zero-sum games* [28, 35] and *Markov decision processes* (MDPs) [26, 4, 36] are two popular frameworks to model decision making in adversarial and uncertain environments respectively. In the former, a system controller and its environment compete antagonistically, and synthesis aims at building strategies for the controller ensuring a specified behavior *against all possible strategies of the environment*. In the latter, the system is faced with a given stochastic model of its environment, and the focus is on satisfying a given level of expected performance, or a

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 8; pp. 8:1–8:18
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

specified behavior with a sufficient probability. Classical objectives studied in both settings include *parity*, a canonical way of encoding ω-regular specifications, and *mean-payoff*, which evaluates the average payoff per transition in the limit of an infinite run in a weighted graph.

Window objectives in games. The traditional parity and mean-payoff objectives share two shortcomings. First, they both reason about infinite runs *in their limit.* While this elegant abstraction yields interesting theoretical properties and makes for robust interpretation, it is often beneficial in practical applications to be able to specify a *parameterized time frame* in which an acceptable behavior should be witnessed. Second, both parity and mean-payoff games belong to $\mathsf{UP} \cap \mathsf{coUP}$ [34, 27], but despite recent breakthroughs [16, 22], they are still not known to be in P. Furthermore, the latest results [21, 25] indicate that all existing algorithmic approaches share inherent limitations that prevent inclusion in P.

Window objectives address the time frame issue as follows. In their *fixed* variant, they consider a window of size bounded by $\lambda \in \mathbb{N}_0$ (given as a parameter) sliding over an infinite run and declare this run to be winning if, in all positions, the window is such that the (mean-payoff or parity) objective is locally satisfied. In their *bounded* variant, the window size is not fixed a priori, but a run is winning if there exists a bound λ for which the condition holds. Window objectives have been considered both in *direct* versions, where the window property must hold from the start of the run, and *prefix-independent* versions, where it must hold from some point on. Window games were initially studied for mean-payoff [17] and parity [14]. They have since seen diverse extensions and applications: e.g., [5, 3, 11, 15, 32, 38].

Window objectives in MDPs. Our goal is to lift the theory of window games to the stochastic context. With that in mind, we consider the canonical *threshold probability problem*: given an MDP, a window objective defining a set of acceptable runs E, and a probability threshold α, we want to decide if there exists a controller *strategy* (also called *policy*) to achieve E with probability at least α. It is well-known that many problems in MDPs can be reduced to threshold problems for appropriate objectives: e.g., maximizing the *expectation* of a prefix-independent function boils down to maximizing the probability to reach the best end-components for that function (see examples in [4, 13, 37]).

Example. Before going further, let us consider an example. Take the MDP depicted in Fig. 1(a): circles depict states and dots depict actions, labeled by letters. Each action yields a probability distribution over successor states: for example, action b leads to s_2 with probability 0.5 and s_3 with the same probability. This MDP is actually a *Markov chain* (MC) as the controller has only one action available in each state: this process is purely stochastic.

We consider the *parity* objective here: each state is associated with a non-negative integer priority and a run is winning if the minimum one amongst those seen infinitely often is even. Clearly, any run in this MC is winning: either it goes through s_3 infinitely often and the min. priority is 0, or it does not, and the min. priority seen infinitely often is 2. Hence the controller not only wins *almost-surely* (with probability one), but even *surely* (on all runs).

Now, consider the *window parity* objective that informally asks for the minimum priority inside a window of size bounded by λ to be even, with this window sliding all along the infinite run. Fix any $\lambda \in \mathbb{N}_0$. It is clear that every time s_1 is visited, there will be a fixed strictly positive probability $\varepsilon > 0$ of not seeing 0 before λ steps: this probability is $1/2^{\lambda-1}$. Let us call this seeing a *bad window.* Since we are in a *bottom strongly connected component* of the MC, we will almost-surely visit s_1 infinitely often [4]. Using classical probability arguments (Borel-Cantelli), one can easily be convinced that the probability to see bad

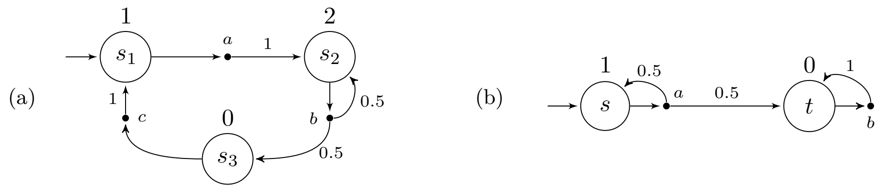


Figure 1 Markov chains where (a) parity is surely satisfied but all window parity objectives have probability zero, and (b) there is no uniform bound over all runs, in contrast to the game setting.

windows infinitely often is one. Hence the probability to win the window parity objective is zero. This canonical example illustrates the difference between traditional parity and window parity: the latter is more restrictive as it asks for a strict bound on the time frame in which each odd priority should be answered by a smaller even priority.

Note that in practice, such a behavior is often wished for. For example, consider a computer server having to grant requests to clients. A classical parity objective can encode that requests should eventually be granted. However, it is clear that in a desired controller, requests should not be placed on hold for an arbitrarily long time. The existence of a finite bound on this holding time can be modeled with a *bounded* window parity objective, while a specific bound can also be set as a parameter using a *fixed* window parity objective.

Our contributions. We study the *threshold probability problem* in MDPs for window objectives based on *parity* and *mean-payoff*, two prominent formalisms in qualitative and quantitative (resp.) analysis of systems. We consider the different variants of window objectives mentioned above: fixed vs. bounded, direct vs. prefix-independent. A nice feature of our approach is that we provide a *unified view* of parity and mean-payoff window objectives: *our algorithm can actually be adapted for any window-based objective* if an appropriate black-box is provided for a restricted sub-problem. This has two advantages: (i) conceptually, our approach permits a *deeper understanding of the essence of the window mechanism*, not biased by technicalities of the specific underlying objective; (ii) our framework can *easily* be *extended* to other objectives for which a window version could be defined. This point is of great practical interest too, as it opens the perspective of a modular, generic software tool suite for window objectives.

For the sake of space, we use acronyms below: DFW for direct fixed window, FW for (prefix-independent) fixed window, DBW for direct bounded window, and BW for (prefix-independent) bounded window. Our main contributions are as follows.

1. We solve DFW MDPs through reductions to safety MDPs over *well-chosen unfoldings*. This results in polynomial-time and pseudo-polynomial-time algorithms for the parity and mean-payoff variants respectively. We prove these complexities to be almost tight (Thm. 5), the most interesting case being the PSPACE-hardness of DFW mean-payoff objectives, even in the case of *acyclic* MDPs.
   We also show that no upper bound can be established on the window size needed to win in general (Ex. 3), *in stark contrast to the two-player games situation* (Rmk. 2).
2. We use similar reductions to prove that *finite memory* suffices in the prefix-independent case (Thm. 4). In this case, we can do better than using unfoldings to solve the problem. We study *end-components* (ECs), the crux for all prefix-independent objectives in MDPs: we show that ECs can be classified based on their *two-player zero-sum game interpretation* (Sect. 5). Using the result on finite memory, we prove that in ECs classified as *good*, almost-sure satisfaction of window objectives can be ensured, whereas it is impossible to

satisfy them with non-zero probability in other ECs (Lem. 10). We also establish tight complexity bounds for this classification problem (Thm. 15). This *EC classification* is (conceptually and complexity-wise) the cornerstone to deal with general MDPs.

3. Our general algorithm is developed in Sect. 6: we prove P-completeness for all prefix-independent variants but for the BW mean-payoff one (Thm. 17), where we show that the problem is in NP ∩ coNP and as hard as mean-payoff games, a canonical "hard" problem for that complexity class.
4. For all variants, we prove *tight memory bounds*: see Thm. 5 for the direct fixed variants, Thm. 17 for the prefix-independent fixed and bounded ones. In all cases, *pure* strategies (i.e., without randomness) suffice.
5. We leave out DBW objectives from our analysis as we show they are not well-behaved. We illustrate their behavior in Sect. 3 and discuss their pitfalls in Sect. 7.

Along the way, we develop side results that help drawing a *line between MDPs and games* w.r.t. window objectives: e.g., the existence of a uniform bound in the bounded case (Rmk. 2).

In the game setting, window objectives are all in polynomial time, except for the BW mean-payoff variant, in NP ∩ coNP. Despite clear differences in behaviors, the situation is almost the same here. The only outlier case is the DFW mean-payoff one, whose complexity rises significantly (Thm. 5): the loss of prefix-independence permits to emulate shortest path problems on MDPs, famously hard to solve efficiently (e.g., [30, 37, 9, 31]). In the almost-sure case, however, DFW mean-payoff MDPs collapse to P (Rmk. 6).

In games, window objectives permit to avoid long-standing NP∩coNP complexity barriers for parity [14] and mean-payoff [17]. Since both are known to be in P for the threshold probability problem in MDPs [20, 37], the main interest of window objectives resides in their *modeling power*. Still, they may turn out to be more efficient in practice too, as polynomial-time algorithms for parity and mean-payoff, based on linear programming, are often forsaken in favor of exponential-time value or strategy iteration ones (e.g., [2]).

Related work. We already mentioned many related articles, hence we only briefly discuss some remaining models here. Window parity games are strongly linked to the concept of *finitary ω-regular games*: see, e.g., [19], or [14] for a complete list of references. The window mechanism can be used to ensure a certain form of (local) guarantee over runs: different techniques have been considered in MDPs, notably *variance-based* [10] or *worst-case-based* [13, 7] methods.

Finally, let us mention the recent work of Bordais et al. [8], considering a *seemingly* related question: the authors define a *value function* based on the window *mean-payoff* mechanism and consider maximizing its expected value (which is different from the expected window size we discuss in Sect. 7). While there are similarities in our works w.r.t. technical tools, the two approaches are quite different and have their own strengths: we focus on deep understanding of the window mechanism through a *generic approach* for the canonical threshold probability problem for all window-based objectives, here instantiated as mean-payoff *and* parity; whereas Bordais et al. focus on a particular *optimization problem* for a function relying on the window mechanism. We mention three examples illustrating the conceptual gap. First, in [8], the studied function takes the same value for direct and prefix-independent bounded window mean-payoff objectives, whereas we show in Sect. 3 that the classical definitions of window objectives induce a striking difference between both (in the MDP of Ex. 3, the prefix-independent version is satisfied for window size one, whereas no uniform bound on all runs can be defined for the direct case). Second, we prove PSPACE-hardness for the DFW mean-payoff case, whereas the best lower bound known for the related problem in [8] is PP. Lastly, recall that we also deal with window *parity* objectives while the function of [8] is strictly built on mean-payoff.

Outline. Sect. 2 defines the problem under study. In Sect. 3, we introduce window objectives, discuss their status in games, and illustrate their behavior in MDPs. Sect. 4 is devoted to the fixed variants and the aforementioned reductions. In Sect. 5, we analyze the case of ECs and develop the classification procedure. We build on it in Sect. 6 to solve the general case. Finally, in Sect. 7, we discuss the limitations of our work, as well as interesting extensions within arm's reach (e.g., multi-objective threshold problem, expected value problem). Full details and proofs can be found in the full version of this paper [12].

2 Preliminaries

Markov decision processes. Given a set S, let $\mathcal{D}(S)$ be the set of rational probability distributions over S. Given $\iota \in \mathcal{D}(S)$, let $\mathsf{Supp}(\iota) = \{s \in S \mid \iota(s) > 0\}$ be its support. A finite *Markov decision process* (MDP) is a tuple $\mathcal{M} = (S, A, \delta)$ where S is a finite set of *states*, A is a finite set of *actions* and $\delta \colon S \times A \to \mathcal{D}(S)$ is a partial function called the *probabilistic transition function.* The set of actions available in $s \in S$ (i.e., for which $\delta(s, a)$ is defined) is $A(s)$. We assume w.l.o.g. that MDPs are *deadlock-free*: for all $s \in S$, $A(s) \neq \emptyset$. An MDP where for all $s \in S$, $|A(s)| = 1$ is a fully-stochastic process called a *Markov chain* (MC).

A *run* of $\mathcal{M}$ is an infinite sequence $\rho = s_0 a_0 \dots a_{n-1} s_n \dots$ of states and actions such that $\delta(s_i, a_i, s_{i+1}) > 0$ for all $i \geq 0$. The *prefix* up to the n-th state of ρ is the finite sequence $\rho[0, n] = s_0 a_0 \dots a_{n-1} s_n$. The *suffix* of ρ starting from the n-th state of ρ is the run $\rho[n, \infty] = s_n a_n s_{n+1} a_{n+1} \dots$. Moreover, we denote by $\rho[n]$ the n-th state s_n of ρ. Finite prefixes of runs of the form $h = s_0 a_0 \dots a_{n-1} s_n$ are called *histories.* We resp. denote the sets of runs and histories of an MDP $\mathcal{M}$ by $\mathsf{Runs}(\mathcal{M})$ and $\mathsf{Hists}(\mathcal{M})$.

Strategies, induced MC and events. A *strategy* σ is a function $\mathsf{Hists}(\mathcal{M}) \to \mathcal{D}(A)$ such that for all $h \in \mathsf{Hists}(\mathcal{M})$ ending in s, we have $\mathsf{Supp}(\sigma(h)) \subseteq A(s)$. The set of all strategies is Σ. A strategy is *pure* if all histories are mapped to *Dirac distributions*, i.e., the support is a singleton. A strategy σ can be encoded by a stochastic state machine with outputs, called *Mealy machine.* We say that σ is *finite-memory* if this machine is finite, and memoryless if it has only one state, i.e., it only depends on the last state of the history. We see such strategies as functions $s \mapsto \mathcal{D}(A(s))$ for $s \in S$. The entity choosing the strategy is called the *controller.*

An MDP $\mathcal{M}$, a strategy σ, and a state s determine a Markov chain $\mathcal{M}_s^\sigma$. When considering the probabilities of events in $\mathcal{M}_s^\sigma$, we will often consider sets of runs of $\mathcal{M}$. Thus, given $E \subseteq (SA)^\omega$, we denote by $\mathbb{P}^\sigma_{\mathcal{M},s}[E]$ the probability of the runs of $\mathcal{M}_s^\sigma$ whose projection to $\mathcal{M}$ is in E, i.e., the probability of event E when $\mathcal{M}$ is executed with initial state s and strategy σ. For the sake of readability, we make similar abuse of notation – identifying runs in the induced MC with their projections in the MDP – throughout our paper.

Note that every measurable set (*event*) has a uniquely defined probability [40] (Carathéodory's extension theorem induces a unique probability measure on the Borel σ-algebra over cylinders of $(SA)^\omega$). When non-ambiguous, we drop some subscripts of $\mathbb{P}^\sigma_{\mathcal{M},s}$.

We say that an event $E \subseteq (SA)^\omega$ is *sure*, written $\mathsf{S}^\sigma_{\mathcal{M},s}[E]$, if and only if $\mathsf{Runs}(\mathcal{M}_s^\sigma) \subseteq E$ and that E is *almost-sure*, written $\mathsf{AS}^\sigma_{\mathcal{M},s}[E]$, if and only if $\mathbb{P}^\sigma_{\mathcal{M},s}[E] = 1$.

Limiting behavior. Fix an MDP $\mathcal{M} = (S, A, \delta)$. A *sub-MDP* of $\mathcal{M}$ is an MDP $\mathcal{M}' = (S', A', \delta')$ with $S' \subseteq S$, $\emptyset \neq A'(s) \subseteq A(s)$ for all $s \in S'$, $\mathsf{Supp}(\delta(s, a)) \subseteq S'$ for all $s \in S', a \in A'(s)$, $\delta' = \delta|_{S' \times A'}$. Such a sub-MDP $\mathcal{M}'$ is an *end-component* (EC) of $\mathcal{M}$ if and only if the underlying graph of $\mathcal{M}'$ is *strongly connected*, i.e., there is a run between any pair of states in S'. Given an EC $\mathcal{M}' = (S', A', \delta')$ of $\mathcal{M}$, we say that its sub-MDP $\mathcal{M}'' = (S'', A'', \delta'')$,

$S'' \subseteq S'$, $A'' \subseteq A'$, is a *sub-EC* of $\mathcal{M}'$ if $\mathcal{M}''$ is also an EC. We let $\mathsf{EC}(\mathcal{M})$ denote the set of ECs of $\mathcal{M}$, which may be of exponential size as ECs need not be disjoint. The counterparts of ECs in MCs are *bottom strongly-connected components* (BSCCs).

The union of two ECs with non-empty intersection is an EC: hence we can define the *maximal* ECs (MECs) of an MDP, i.e., the ECs that cannot be extended. We let $\mathsf{MEC}(\mathcal{M})$ denote the set of MECs of $\mathcal{M}$, of polynomial size (because MECs are pair-wise disjoint) and computable in polynomial time [18].

Given some run $\rho = s_0a_0s_1a_1\ldots \in \mathsf{Runs}(\mathcal{M})$, let $\mathsf{inf}(\rho) = \{s \in S \mid \forall\, i \geq 0,\, \exists\, j > i,\, s_j = s\}$ denote the states visited infinitely-often along ρ, and let $\mathsf{infAct}(\rho) = \{a \in A \mid \forall\, i \geq 0,\, \exists\, j > i,\, a_j = a\}$ similarly denote the actions taken infinitely-often along ρ. Let $\mathsf{limitSet}(\rho)$ denote the pair $(\mathsf{inf}(\rho), \mathsf{infAct}(\rho))$. Note that this pair may induce a well-defined sub-MDP $\mathcal{M}' = (\mathsf{inf}(\rho), \mathsf{infAct}(\rho), \delta|_{\mathsf{inf}(\rho)\times\mathsf{infAct}(\rho)})$, but in general it need not be the case. A folklore result in MDPs (e.g., [4]) is the following: for any state s of MDP $\mathcal{M}$, for any strategy $\sigma \in \Sigma$, we have that $\mathsf{AS}^{\sigma}_{\mathcal{M},s}[\{\rho \in \mathsf{Runs}(\mathcal{M}^{\sigma}_{s}) \mid \mathsf{limitSet}(\rho) \in \mathsf{EC}(\mathcal{M})\}]$, that is, under any strategy, the limit behavior of the MDP almost-surely coincides with an EC. This property is a key tool in the analysis of MDPs with prefix-independent objectives, as it essentially says that we only need to identify the "best" ECs and maximize the probability to reach them.

Weights, priorities and complexity. In this paper, we always assume an MDP with either (i) a *weight* function $w\colon A \to \mathbb{Z}$ of largest absolute weight W, or (ii) a *priority* function $p\colon S \to \{0, 1, \ldots, d\}$, with $d \leq |S| + 1$ (w.l.o.g.). This choice is left implicit when the context is clear, to offer a unified view of mean-payoff and parity variants of window objectives.

Regarding complexity, we make the classical assumptions of the field: we consider the model size $|\mathcal{M}|$ to be polynomial in $|S|$ and the *binary encoding* of weights and probabilities (e.g., $V = \log_2 W$), whereas we consider the largest priority d, as well as the upcoming window size λ, to be encoded in unary. When a problem is polynomial in W, we say that it is *pseudo*-polynomial: it would be polynomial if weights would be given in unary.

Objectives. An *objective* for an MDP $\mathcal{M} = (S, A, \delta)$ is a measurable set of runs $E \subseteq (SA)^{\omega}$. We consider window objectives based on *mean-payoff* and *parity* objectives. Let us discuss those classical objectives.

- *Mean-Payoff* is a *quantitative* objective for which we consider *weighted* MDPs. Let $\rho \in \mathsf{Runs}(\mathcal{M})$ be a run of such an MDP. The *mean-payoff* of prefix $\rho[0, n]$ is $\mathsf{MP}(\rho[0, n]) = \frac{1}{n}\sum_{i=0}^{n-1} w(a_i)$, for $n > 0$. This is naturally extended to runs by considering the limit behavior. The *mean-payoff* of ρ is $\mathsf{MP}(\rho) = \liminf_{n\to\infty} \mathsf{MP}(\rho[0, n])$. Given a threshold $\nu \in \mathbb{Q}$, the mean-payoff objective accepts all runs whose mean-payoff is above the threshold, i.e., $\mathsf{MeanPayoff}(\nu) = \{\rho \in \mathsf{Runs}(\mathcal{M}) \mid \mathsf{MP}(\rho) \geq \nu\}$. Note that ν can be taken equal to zero w.l.o.g., and the mean-payoff function can be equivalently (in the classical one-dimension setting) defined using $\limsup$.
- *Parity* is a *qualitative* objective for which we consider MDPs with a priority function. It requires that the smallest priority seen infinitely often along a run be even, i.e., $\mathsf{Parity} = \{\rho \in \mathsf{Runs}(\mathcal{M}) \mid \min_{s\in\mathsf{inf}(\rho)} p(s) = 0 \pmod 2\}$.

Decision problem. Given an MDP $\mathcal{M} = (S, A, \delta)$, an initial state s, a threshold $\alpha \in [0, 1]\cap\mathbb{Q}$, and an objective E, the *threshold probability problem* is to decide whether there exists a strategy $\sigma \in \Sigma$ such that $\mathbb{P}^{\sigma}_{\mathcal{M},s}[E] \geq \alpha$ or not.

Furthermore, if it exists, we want to build such a strategy. The related problems for both mean-payoff and parity are in P and pure memoryless strategies suffice [37, 20].

3 Window objectives

Good Windows. Given a weighted MDP $\mathcal{M}$ and $\lambda > 0$, we define the *good window mean-payoff* objective $\mathsf{GW}_{\mathsf{mp}}(\lambda) = \big\{\rho \in \mathsf{Runs}(\mathcal{M}) \mid \exists\, l < \lambda,\ \mathsf{MP}\big(\rho[0, l+1]\big) \geq 0\big\}$, requiring the existence of a window of size bounded by λ and starting at the first position of the run, over which the mean-payoff is at least equal to zero (w.l.o.g.).

Similarly, given an MDP $\mathcal{M}$ with priority function p, we define the *good window parity* objective, $\mathsf{GW}_{\mathsf{par}}(\lambda) = \big\{\rho \in \mathsf{Runs}(\mathcal{M}) \mid \exists\, l < \lambda,\ \big(p(\rho[l]) \bmod 2 = 0 \wedge \forall\, k < l,\ p(\rho[l]) < p(\rho[k])\big)\big\}$, requiring the existence of a window of size bounded by λ and starting at the first position of the run, for which the last priority is even and is the smallest within the window.

We use subscripts mp and par for mean-payoff and parity variants respectively. So, given $\Omega = \{\mathsf{mp}, \mathsf{par}\}$ and a run $\rho \in \mathsf{Runs}(\mathcal{M})$, we say that *an Ω-window is closed* in at most λ steps from $\rho[i]$ if $\rho[i, \infty]$ is in $\mathsf{GW}_{\Omega}(\lambda)$. If a window is not yet closed, we call it *open*.

Fixed variants. Given $\lambda > 0$, we define the *direct fixed window* objective $\mathsf{DFW}_{\Omega}(\lambda) = \{\rho \in \mathsf{Runs}(\mathcal{M}) \mid \forall\, j \geq 0,\ \rho[j, \infty] \in \mathsf{GW}_{\Omega}(\lambda)\}$, asking for all Ω-windows to be closed within λ steps along the run. We also define the *fixed window* objective $\mathsf{FW}_{\Omega}(\lambda) = \{\rho \in \mathsf{Runs}(\mathcal{M}) \mid \exists\, i \geq 0,\ \ \rho[i, \infty] \in \mathsf{DFW}_{\Omega}(\lambda)\}$, that is the *prefix-independent* version of the previous one: it requires it to be eventually satisfied.

Bounded variant. The *bounded window* objective $\mathsf{BW}_{\Omega} = \{\rho \in \mathsf{Runs}(\mathcal{M}) \mid \exists\, \lambda > 0,\ \ \rho \in \mathsf{FW}_{\Omega}(\lambda)\}$, requires the existence of a λ for which the fixed window objective is satisfied. Note that this bound need not be *uniform* along all runs in general. A *direct* variant may also be defined, but turns out to be ill-suited in the stochastic context: we illustrate it in Ex. 3 and discuss its pitfalls in Sect. 7. Hence we focus on the prefix-independent version here.

▶ **Example 1.** We first go back to the example of Sect. 1, depicted in Fig. 1(a). Let $\mathcal{M}$ be this MDP. Fix run $\rho = (s_1\, a\, s_2\, b\, s_3\, c)^{\omega}$. We have that $\rho \notin \mathsf{FW}_{\mathsf{par}}(\lambda = 2)$ – a fortiori, $\rho \notin \mathsf{DFW}_{\mathsf{par}}(\lambda = 2)$ – as the window that opens in s_1 is not closed after two steps (because s_1 has odd priority 1, and 2 is not smaller than 1 so does not suffice to answer it). If we now set $\lambda = 3$, we see that this window closes on time, as 0 is encountered within three steps. As all other windows are immediately closed, we have $\rho \in \mathsf{DFW}_{\mathsf{par}}(\lambda = 3)$ – a fortiori, $\rho \in \mathsf{FW}_{\mathsf{par}}(\lambda = 3)$ and $\rho \in \mathsf{BW}_{\mathsf{par}}$.

Regarding the probability of these objectives, however, we have already argued that, for all $\lambda > 0$, $\mathbb{P}_{\mathcal{M}, s_1}[\mathsf{FW}_{\mathsf{par}}(\lambda)] = 0$, whereas $\mathbb{P}_{\mathcal{M}, s_1}[\mathsf{Parity}] = 1$ since s_3 is almost-surely visited infinitely often but any time bound is almost-surely exceeded infinitely often too. Observe that $\mathsf{BW}_{\mathsf{par}} = \bigcup_{\lambda > 0} \mathsf{FW}_{\mathsf{par}}(\lambda)$, hence we also have that $\mathbb{P}_{\mathcal{M}, s_1}[\mathsf{BW}_{\mathsf{par}}] = 0$ (by countable additivity). Similar reasoning holds for window mean-payoff, by taking the weight function $w = \{a \mapsto -1,\, b \mapsto 0, c \mapsto 1\}$.

▶ Remark 2. Window mean-payoff and window parity objectives were considered in two-player zero-sum games [17, 14]. This setting is equivalent to deciding if there exists a strategy in an MDP such that the objective is *surely* satisfied, i.e., by all consistent runs. Interestingly, in games, if the controller can win the bounded window objective, then a uniform bound exists, i.e., there exists a window size λ sufficiently large such that the bounded version coincides with the fixed one. Recall that this is not granted by definition. This uniform bound is pseudo-polynomial for mean-payoff and equal to the number of states for parity. We illustrate in the following example that *in MDPs, a uniform bound need not exist.*

▶ **Example 3.** Consider the MC $\mathcal{M}$ in Fig. 1(b). For any $\lambda > 0$, there is probability $1/2^{\lambda-1}$ that objective $\mathsf{DFW}_{\mathsf{par}}(\lambda)$ is not satisfied. Hence, for all $\lambda > 0$, $\mathbb{P}_{\mathcal{M},s}[\mathsf{DFW}_{\mathsf{par}}(\lambda)] < 1$.

Now let $\mathsf{DBW}_{\mathsf{par}} = \bigcup_{\lambda > 0} \mathsf{DFW}_{\mathsf{par}}(\lambda)$ be the *direct* bounded window objective evoked above. We claim that $\mathbb{P}_{\mathcal{M},s}[\mathsf{DBW}_{\mathsf{par}}] = 1$. Indeed, any run ending in t belongs to $\mathsf{DBW}_{\mathsf{par}}$, as it belongs to $\mathsf{DFW}_{\mathsf{par}}(\lambda)$ for λ equal to the length of the prefix up to t. Since t is almost-surely reached (as it is the only BSCC of the MC), we conclude that $\mathsf{DBW}_{\mathsf{par}}$ is indeed satisfied almost-surely.

Essentially, the difference stems from the fairness of probabilities. In a game, the opponent controls the successor choice after action a and always goes back to s, resulting in objective $\mathsf{DBW}_{\mathsf{par}}$ being lost. However, in this MC, s is almost-surely left eventually, but we cannot guarantee when: hence there exists a window bound for each run, but there is no uniform bound over all runs.

This MC also illustrates the difference between sure and almost-sure satisfaction: we have $\mathsf{AS}_{\mathcal{M},s}[\mathsf{FW}_{\mathsf{par}}(\lambda = 1)]$ but not $\mathsf{S}_{\mathcal{M},s}[\mathsf{FW}_{\mathsf{par}}(\lambda = 1)]$, because of run $\rho = (s\, a)^{\omega}$. Again the same reasoning holds for mean-payoff variants, for example with $w = \{a \mapsto -1, b \mapsto 1\}$.

We leave out the *direct* bounded objective from now on. We will come back to it in Sect. 7 and motivate why this objective is not well-behaved. In the following, we focus on direct and prefix-independent fixed window objectives and prefix-independent bounded ones.

4 Fixed case: better safe than sorry

We start with the fixed variants of window objectives. Our main goal here is to establish that *pure finite-memory* strategies suffice in all cases. As a by-product, we also obtain algorithms to solve the corresponding decision problems. Still, for the prefix-independent variants, we will obtain better complexities using the upcoming generic approach (Sect. 6).

Our tools are natural reductions from direct (resp. prefix-independent) window problems on MDPs to safety (resp. co-Büchi) problems on *unfoldings* based on the window size λ (i.e., larger arenas incorporating information on open windows).

Unfoldings. We use *identical unfoldings* for both direct and prefix-independent objectives. Let $\mathcal{M}$ be an MDP and $\lambda > 0$ be the window size. We build a new MDP $\mathcal{M}_{\lambda}$, the unfolding of $\mathcal{M}$ for mean-payoff (resp. parity), with an extended state space $\tilde{S}$: each state of $\mathcal{M}_{\lambda}$ is of the form $\tilde{s} = (s, l, x)$ so that s keeps track of the current state of $\mathcal{M}$, l of the size of the current open window, and x of the current sum of weights (resp. the minimum priority) in the window: the last two values are therefore reset whenever a window is closed or stays open for λ steps. The remaining components of $\mathcal{M}_{\lambda}$ are then extended from $\mathcal{M}$ in a natural way.

A key underlying property used here is the so-called *inductive property of windows* [17, 14]: for all runs $\rho = s_0 a_0 s_1 a_1 \ldots$ of $\mathcal{M}$, fix a window starting in position $i \geq 0$ and let j be the position in which this window gets closed, assuming it does. Then, all windows in positions from i to j also close in j. The validity of this property is easy to check by contradiction (if it would not hold, the window in i would close before j). This property is fundamental in our reduction: without it we would have to keep track of all open windows in parallel, which would result in a blow-up exponential in λ.

Reductions. A safety (resp. co-Büchi) objective consists of runs avoiding at all times (resp. eventually avoiding) a given set of states B. In $\mathcal{M}_{\lambda}$, B is composed of states (s, l, x) where $l = \lambda$ and $x < 0$ for the mean-payoff variant or $l = \lambda - 1$ and $x \bmod 2 = 1$ for the parity variant, that exactly correspond to windows staying open for λ steps.

We state that the safety and co-Büchi objectives in $\mathcal{M}_\lambda$ are *probability-wise equivalent* to the direct fixed window and fixed window ones in $\mathcal{M}$. For any strategy σ in $\mathcal{M}$, $s \in S$ and its corresponding state $\tilde{s} \in \tilde{S}$, there exists a strategy $\tilde{\sigma}$ in $\mathcal{M}_\lambda$ such that the probability of satisfying the direct fixed (resp. fixed) window objective of maximal window size λ in $\mathcal{M}^\sigma_s$ equals the probability of satisfying the safety (resp. co-Büchi) objective in $\mathcal{M}^{\tilde{\sigma}}_{\lambda,\tilde{s}}$, and conversely. To obtain a strategy σ in $\mathcal{M}$ from a strategy $\tilde{\sigma}$ in $\mathcal{M}_\lambda$, we have to integrate in the memory of σ the additional information encoded in $\tilde{S}$: hence the memory required by σ is the one used by $\tilde{\sigma}$ with a blow-up polynomial in $|\tilde{S}|$. These reductions, along with the fact that pure memoryless strategies suffice for safety and co-Büchi objectives in MDPs [4], yield sufficiency of finite-memory strategies, a *key ingredient* in our generic approach (Lem. 10).

▶ **Theorem 4.** *Pure finite-memory strategies suffice for the threshold probability problem for all fixed window objectives. That is, given MDP $\mathcal{M} = (S, A, \delta)$, initial state $s \in S$, window size $\lambda > 0$, $\Omega \in \{\mathsf{mp}, \mathsf{par}\}$, objective $E \in \{\mathsf{DFW}_\Omega(\lambda), \mathsf{FW}_\Omega(\lambda)\}$ and threshold probability $\alpha \in [0, 1] \cap \mathbb{Q}$, if there exists a strategy $\sigma \in \Sigma$ such that $\mathbb{P}^\sigma_{\mathcal{M},s}[E] \geq \alpha$, then there exists a pure finite-memory strategy σ' such that $\mathbb{P}^{\sigma'}_{\mathcal{M},s}[E] \geq \alpha$.*

These reductions also yield *algorithms* for the fixed window case. We only use them for the direct variants, as the approach we develop in Sect. 6 proves to be more efficient for the prefix-independent one, for two reasons: first, we may restrict the co-Büchi-like analysis to ECs; second, we use a more tractable analysis than the co-Büchi unfolding for mean-payoff.

We complement the corresponding upper bounds with almost-matching lower bounds, showing that our approach is close to optimal, complexity-wise.

▶ **Theorem 5.** *The threshold probability problem is*

(a) P-*complete for direct fixed window parity objectives, and pure polynomial-memory optimal strategies can be constructed in polynomial time. Furthermore, polynomial memory is in general necessary.*

(b) *in* $\mathsf{EXPTIME}$ *and* PSPACE-*hard for direct fixed window mean-payoff objectives (already for acyclic MDPs), and pure pseudo-polynomial-memory optimal strategies can be constructed in pseudo-polynomial time. Furthermore, pseudo-polynomial memory is in general necessary.*

Upper bounds. The algorithm is simple: given $\mathcal{M}$ and $\lambda > 0$, build $\mathcal{M}_\lambda$ and solve the corresponding safety problem. This can be done in polynomial time in $|\mathcal{M}_\lambda|$ and pure memoryless strategies suffice over $\mathcal{M}_\lambda$ [4]. For parity, $\mathcal{M}_\lambda$ is of size polynomial in $|\mathcal{M}|$, d and λ. Since both d (anyway bounded by $\mathcal{O}(|S|)$) and λ are assumed to be given in unary, it yields the result. For mean-payoff, $\mathcal{M}_\lambda$ is of size polynomial in $|\mathcal{M}|$, W and λ. Since weights are assumed to be encoded in binary, we only have a pseudo-polynomial-time algorithm.

Lower bounds. We give some insights about the reductions yielding the results for lower complexity bounds. We begin with **P-hardness** (item (a)). Roughly, we reduce two-player reachability games [6, 33] to direct fixed window parity MDPs, using two key ingredients: (i) if winning is possible in the game, it is possible in bounded time: we deduce a sufficient window size λ from it; (ii) *almost-sure* winning for $\mathsf{DFW}_\Omega(\lambda)$ objectives is equivalent to *sure* winning (if a losing run exists, it is witnessed by a finite prefix of strictly positive probability).

Regarding memory, the proof established for direct fixed window parity games in [14] carries over easily to our setting by replacing the states of the opponent by stochastic actions, in the natural way. Hence the lower bound is trivial to establish.

Consider now **PSPACE-hardness** (item (b)). We proceed via a reduction from the threshold probability problem for shortest path objectives [30, 37]. Given an MDP $\mathcal{M}$ with state space S, a lower probability bound α and an upper bound $\ell \in \mathbb{N}$ on the cumulative sum of weights of actions through runs of the system, this problem asks whether there exists a strategy allowing to visit a target set $T \subseteq S$ with probability at least α and cost at most ℓ (note that weights are assumed to be strictly positive in this setting). The problem is known to be PSPACE-hard, even for *acyclic* MDPs [30].

We establish a reduction from this problem, in the acyclic case, to a threshold probability problem for $\mathsf{DFW}_{\mathsf{mp}}(\lambda = |S|)$, maintaining the acyclicity of the underlying graph. From $\mathcal{M}$, we build a new MDP $\mathcal{M}'$ by taking the opposite of all weights; adding the bound ℓ when entering the target; and making the target cost-free. The result follows from three key ingredients: (i) the sum of weights over a prefix in $\mathcal{M}'$ that is not yet in T is strictly negative, and the opposite of the sum over the same prefix in $\mathcal{M}$; (ii) due to the addition of ℓ on entering T, any run of $\mathcal{M}'$ sees all its windows closed if and only if T is reached with a cost less than ℓ in $\mathcal{M}$; (iii) using the acyclicity, if a run reaches T, it does so within λ steps.

The need for pseudo-polynomial memory is also proved through this reduction. Indeed, there is a chain of reductions from *subset-sum games* [39, 24] to our setting, via the shortest path problem [30]. Subset-sum games require pseudo-polynomial-memory strategies.

▶ Remark 6. As noted above, almost-surely winning coincides with surely winning for the *direct fixed* window objectives. Therefore, the threshold probability problem for $\mathsf{DFW}_{\mathsf{mp}}(\lambda)$ collapses to P if $\alpha = 1$ [17].

5 The case of end-components

We have solved the case of direct fixed window objectives: it remains to consider prefix-independent fixed and bounded variants. The analysis of MDPs with prefix-independent objectives crucially relies on ECs (Sect. 2): they are almost-surely reached in the long run.

First, we study what happens in ECs: how to play optimally and what can be achieved. In Sect. 6, we will use this knowledge as the cornerstone of our algorithm for general MDPs. The main result here is a *strong link between ECs and two-player games*: intuitively, either the probability to win a window objective in an EC is zero, or it is one and there exists a sub-EC where the controller can actually win surely, i.e., as in a two-player game played on this sub-EC. We start by defining the notion of λ-safety, that will characterize such sub-ECs.

▶ **Definition 7** (λ-safety). *Let $\mathcal{M}$ be an MDP, $\Omega \in \{\mathsf{mp}, \mathsf{par}\}$, $\lambda > 0$, and $\mathcal{C} = (S_{\mathcal{C}}, A_{\mathcal{C}}, \delta_{\mathcal{C}}) \in \mathsf{EC}(\mathcal{M})$, we say that $\mathcal{C}$ is λ-safe$_\Omega$ if there exists a strategy $\sigma \in \Sigma$ in $\mathcal{C}$ such that, from all $s \in S_{\mathcal{C}}$, $\mathsf{S}^{\sigma}_{\mathcal{C},s}[\mathsf{DFW}_\Omega(\lambda)]$.*

Classifying an EC as λ-safe$_\Omega$ or not boils down to interpreting it as a *two-player game* (the duality between MDPs and games is further explored in [13, 7]). The uncertainty becomes adversarial: on entering a state s of the MDP, the controller chooses an action a following its strategy and the opponent then chooses a successor s' such that $s' \in \mathsf{Supp}(\delta(s, a))$ without taking into account the exact values of probabilities. In such a view, the opponent tries to prevent the controller from achieving its objective. A *winning strategy* for the controller in the game interpretation is a strategy that ensures the objective regardless of its opponent's strategy. An EC is thus said to be λ-safe$_\Omega$ if and only if its two-player interpretation admits a winning strategy for $\mathsf{DFW}_\Omega(\lambda)$. W.l.o.g., all our strategies are uniform in the game-theoretic sense: we use it in our statements.

▶ **Proposition 8.** *Let $\mathcal{M}$ be an MDP, $\Omega \in \{\mathsf{mp}, \mathsf{par}\}$, $\lambda > 0$, and $\mathcal{C} = (S_\mathcal{C}, A_\mathcal{C}, \delta_\mathcal{C}) \in \mathsf{EC}(\mathcal{M})$ be λ-safe$_\Omega$. Then, there exists a pure polynomial-memory strategy $\sigma_{safe}^{\Omega,\lambda,\mathcal{C}}$ in $\mathcal{C}$ such that $\mathsf{S}_{\mathcal{C},s}^{\sigma_{safe}^{\Omega,\lambda,\mathcal{C}}}[\mathsf{DFW}_\Omega(\lambda)]$ for all $s \in S_\mathcal{C}$.*

The proof is straightforward by definition of λ-safety and pure polynomial-memory strategies being sufficient in direct fixed window games, both for mean-payoff [17] and parity [14].

As sketched above, the existence of sub-ECs that are λ-safe is crucial in order to satisfy any window objective in an EC. We thus introduce the notion of *good* ECs.

▶ **Definition 9.** *Let $\mathcal{M}$ be an MDP, $\Omega \in \{\mathsf{mp}, \mathsf{par}\}$, and $\mathcal{C} \in \mathsf{EC}(\mathcal{M})$, we say that*

- *$\mathcal{C}$ is λ-good$_\Omega$, for $\lambda > 0$, if it contains a sub-EC $\mathcal{C}'$ which is λ-safe$_\Omega$.*
- *$\mathcal{C}$ is BW-good$_\Omega$ if it contains a sub-EC $\mathcal{C}'$ which is λ-safe$_\Omega$ for some $\lambda > 0$.*

Any BW-good$_\Omega$ EC is λ-good$_\Omega$ for an appropriate $\lambda > 0$. Yet, we use a different terminology as in the BW case, we do not fix λ a priori: this is important complexity-wise.

We now establish that good$_\Omega$ ECs are exactly the ones where window objectives can be satisfied with non-zero probability, and actually, with probability one.

▶ **Lemma 10** (Zero-one law)**.** *Let $\mathcal{M}$ be an MDP, $\Omega \in \{\mathsf{mp}, \mathsf{par}\}$ and $\mathcal{C} = (S_\mathcal{C}, A_\mathcal{C}, \delta_\mathcal{C}) \in \mathsf{EC}(\mathcal{M})$. The following assertions hold.*

(a) *For all $\lambda > 0$,*

(i) *either $\mathcal{C}$ is λ-good$_\Omega$ and there exists a strategy σ in $\mathcal{C}$ such that $\mathsf{AS}_{\mathcal{C},s}^{\sigma}[\mathsf{FW}_\Omega(\lambda)]$ for all $s \in S_\mathcal{C}$,*

(ii) *or for all $s \in S_\mathcal{C}$ and for all strategy σ in $\mathcal{C}$, $\mathbb{P}_{\mathcal{C},s}^{\sigma}[\mathsf{FW}_\Omega(\lambda)] = 0$.*

(b) **(i)** *Either $\mathcal{C}$ is BW-good$_\Omega$ and there exists a strategy σ in $\mathcal{C}$ such that $\mathsf{AS}_{\mathcal{C},s}^{\sigma}[\mathsf{BW}_\Omega]$ for all $s \in S_\mathcal{C}$, or*

(ii) *for all $s \in S_\mathcal{C}$, for all strategy σ in $\mathcal{C}$, $\mathbb{P}_{\mathcal{C},s}^{\sigma}[\mathsf{BW}_\Omega] = 0$.*

We sketch the proof by focusing on case (a). Roughly, (i) holds thanks to the two following facts. First, $\mathcal{C}$ is an EC in which there exists a strategy almost-surely visiting all states of its state space. Second, there is a λ-safe$_\Omega$ sub-EC in $\mathcal{C}$ in which there exists a strategy surely satisfying $\mathsf{DFW}_\Omega(\lambda)$. Combining these two strategies yields the result. For case (ii), recall that finite-memory strategies suffice for $\mathsf{FW}_\Omega(\lambda)$ objectives by Thm. 4. Hence, we fix a finite-memory strategy in $\mathcal{C}$, yielding a finite induced MC where runs almost-surely end up in a BSCC $\mathcal{B}$ [4]. There is no λ-safe$_\Omega$ sub-EC, so there exists a run $\widehat{\rho}$ in $\mathcal{B}$ such that $\widehat{\rho} \notin \mathsf{DFW}_\Omega(\lambda)$. From $\widehat{\rho}$, we extract a history $\widehat{h}$ that contains a window open for λ steps. Since all states in $\mathcal{B}$ are almost-surely visited infinitely often, $\hat{h}$ also happens infinitely often with probability one and the probability to win $\mathsf{FW}_\Omega(\lambda)$ when reaching $\mathcal{B}$ is thus zero. Since this holds for any BSCC induced by σ, we obtain the claim.

Items (b)(i) and (b)(ii) can then be shown by using the fact that $\mathsf{BW}_\Omega = \bigcup_{\lambda > 0} \mathsf{FW}_\Omega(\lambda)$.

▶ Remark 11. An interesting consequence of Lem. 10 is the existence of uniform bounds on λ in ECs, in contrast to the general MDP case, as seen in Sect. 3. This is indeed natural, as we established that winning with positive probability within an EC coincides with winning surely in a sub-EC; sub-EC that can be seen as a two-player zero-sum game where uniform bounds are granted by [17, 14].

Lem. 10 establishes that interesting strategies exist in good$_\Omega$ ECs. Let us describe them.

▶ **Proposition 12.** *Let $\mathcal{M}$ be an MDP, $\Omega \in \{\mathsf{mp}, \mathsf{par}\}$, and $\mathcal{C} = (S_\mathcal{C}, A_\mathcal{C}, \delta_\mathcal{C}) \in \mathsf{EC}(\mathcal{M})$.*

- *If $\mathcal{C}$ is λ-good$_\Omega$, for $\lambda > 0$, then there exists a pure polynomial-memory strategy $\sigma_{good}^{\Omega,\lambda,\mathcal{C}}$ such that $\mathsf{AS}_{\mathcal{C},s}^{\sigma_{good}^{\Omega,\lambda,\mathcal{C}}}[\mathsf{FW}_\Omega(\lambda)]$ for all $s \in S_\mathcal{C}$.*

- *If $\mathcal{C}$ is* BW*-good*$_\Omega$*, then there exists a pure memoryless strategy* $\sigma_{good}^{\Omega,\mathsf{BW},\mathcal{C}}$ *such that* $\mathsf{AS}_{\mathcal{C},s}^{\sigma_{good}^{\Omega,\mathsf{BW},\mathcal{C}}}[\mathsf{BW}_\Omega]$ *for all* $s \in S_\mathcal{C}$*.*

Intuitively, such strategies first mimic a pure memoryless strategy reaching a safe$_\Omega$ sub-EC almost-surely, then switch to a strategy surely winning in this sub-EC, which is lifted from the game interpretation.

We may already sketch a general solution to the threshold probability problem based on Lem. 10 and the well-known fact that ECs are almost-surely reached under any strategy: an optimal strategy must maximize the probability to reach good$_\Omega$ ECs. It is therefore crucial to be able to identify such ECs efficiently. However, an MDP may contain an exponential number of ECs. Fortunately, the next lemma states that we do not have to test them all.

▶ **Lemma 13.** *Let $\mathcal{M}$ be an MDP and $\mathcal{C} \in \mathsf{EC}(\mathcal{M})$. If $\mathcal{C}$ is λ-good$_\Omega$ (resp.* BW*-good*$_\Omega$*), then it is also the case of any super-EC $\mathcal{C}' \in \mathsf{EC}(\mathcal{M})$ containing $\mathcal{C}$.*

▶ **Corollary 14.** *Let $\mathcal{M}$ be an MDP and $\mathcal{C} \in \mathsf{MEC}(\mathcal{M})$ be a maximal EC. If $\mathcal{C}$ is not λ-good$_\Omega$ (resp.* BW*-good*$_\Omega$*), then neither is any of its sub-EC $\mathcal{C}' \in \mathsf{EC}(\mathcal{M})$.*

The interest of Cor. 14 is that the number of MECs is bounded by $|S|$ for any MDP $\mathcal{M} = (S, A, \delta)$ because they are all disjoint. Furthermore, decomposing $\mathcal{M}$ in MECs can be done efficiently (e.g., quadratic time [18]). So, we know classifying MECs is sufficient and MECs can easily be identified: it remains to discuss how to classify a MEC as good$_\Omega$ or not.

Let $\mathcal{M} = (S, A, \delta)$. Recall that a MEC $\mathcal{C} = (S_\mathcal{C}, A_\mathcal{C}, \delta_\mathcal{C}) \in \mathsf{MEC}(\mathcal{M})$ is λ-good$_\Omega$ (resp. BW-good$_\Omega$) if and only if it contains a λ-safe$_\Omega$ sub-EC. This is equivalent to having a non-empty *winning set* for the controller in the two-player game over $\mathcal{C}$ – naturally defined as above. This winning set contains all states in $S_\mathcal{C}$ from which the controller has a *surely winning* strategy. This set, if non-empty, contains at least one sub-EC of $\mathcal{C}$, as otherwise the opponent could force the controller to leave it and win the game (by prefix-independence). Thus, testing if a MEC is good$_\Omega$ boils down to solving its two-player game interpretation [17, 14].

▶ **Theorem 15** (MEC classification)**.** *Let $\mathcal{M}$ be an MDP and $\mathcal{C} \in \mathsf{MEC}(\mathcal{M})$. The following assertions hold.*

(a) *Deciding if $\mathcal{C}$ is λ-good$_\Omega$, for $\lambda > 0$, is in* P *for $\Omega \in \{\mathsf{mp}, \mathsf{par}\}$ and a corresponding pure polynomial-memory strategy $\sigma_{good}^{\Omega,\lambda,\mathcal{C}}$ can be constructed in polynomial time.*

(b) *Deciding if $\mathcal{C}$ is* BW*-good*$_{\mathsf{mp}}$ *is in* $\mathsf{NP} \cap \mathsf{coNP}$ *and a corresponding pure memoryless strategy $\sigma_{good}^{\mathsf{mp},\mathsf{BW},\mathcal{C}}$ can be constructed in pseudo-polynomial time.*

(c) *Deciding if $\mathcal{C}$ is* BW*-good*$_{\mathsf{par}}$ *is in* P *and a corresponding pure memoryless strategy $\sigma_{good}^{\mathsf{par},\mathsf{BW},\mathcal{C}}$ can be constructed in polynomial time.*

6 General MDPs

We have all the ingredients to establish an algorithm in the general case. Given an MDP $\mathcal{M}$, a state s and a window objective $\mathsf{FW}_\Omega(\lambda)$ for $\lambda > 0$ (resp. BW_Ω), we (i) compute the MEC decomposition of $\mathcal{M}$; (ii) classify each MEC as λ-good$_\Omega$ (resp. BW-good$_\Omega$) or not; and (iii) compute an optimal strategy from s to reach the union of good$_\Omega$ MECs: the probability of reaching such MECs is exactly the maximum probability for the window objective.

The fixed and bounded versions are presented in Fig. 2. Sub-procedure $\mathsf{MaxReachability}(s, T)$ computes the maximum probability to reach the set T from s (in polynomial time [4]). The overall complexity of the algorithm is dominated by the classification step.

Algorithm 1 $\mathsf{FixedWindow}(\mathcal{M}, s, \Omega, \lambda)$.

```
Input: MDP $\mathcal{M}$, state $s$, $\Omega \in \{\mathsf{mp}, \mathsf{par}\}$, $\lambda > 0$
Output: Maximum probability of $\mathsf{FW}_\Omega(\lambda)$ from $s$
$T \leftarrow \emptyset$
for all $\mathcal{C} = (S_\mathcal{C}, A_\mathcal{C}, \delta_\mathcal{C}) \in \mathsf{MEC}(\mathcal{M})$ do
    if $\mathcal{C}$ is $\lambda$-good$_\Omega$ then
        $T \leftarrow T \uplus S_\mathcal{C}$
$\nu = \mathsf{MaxReachability}(s, T)$
return $\nu$
```

Algorithm 2 $\mathsf{BoundedWindow}(\mathcal{M}, s, \Omega)$.

```
Input: MDP $\mathcal{M}$, state $s$, $\Omega \in \{\mathsf{mp}, \mathsf{par}\}$
Output: Maximum probability of $\mathsf{BW}_\Omega$ from $s$
$T \leftarrow \emptyset$
for all $\mathcal{C} = (S_\mathcal{C}, A_\mathcal{C}, \delta_\mathcal{C}) \in \mathsf{MEC}(\mathcal{M})$ do
    if $\mathcal{C}$ is BW-good$_\Omega$ then
        $T \leftarrow T \uplus S_\mathcal{C}$
$\nu = \mathsf{MaxReachability}(s, T)$
return $\nu$
```

Figure 2 Algorithms computing the max. probability of prefix independent window objectives.

▶ **Lemma 16.** *Alg. 1 and Alg. 2 are correct: given an MDP $\mathcal{M} = (S, A, \delta)$, an initial state $s \in S$, $\Omega \in \{\mathsf{mp}, \mathsf{par}\}$, $\lambda > 0$, we have that* $\mathsf{FixedWindow}(\mathcal{M}, s, \Omega, \lambda) = \max_{\sigma\in\Sigma} \mathbb{P}^\sigma_{\mathcal{M},s}[\mathsf{FW}_\Omega(\lambda)]$ *and* $\mathsf{BoundedWindow}(\mathcal{M}, s, \Omega) = \max_{\sigma\in\Sigma} \mathbb{P}^\sigma_{\mathcal{M},s}[\mathsf{BW}_\Omega]$.

The proof is straightforward based on our previous results: (i) using prefix-independence and almost-sure reachability of MECs, we know that the highest probability (of satisfying the objective) is obtained when the "best" MECs are reached; (ii) by Lem. 10, this probability is one in good$_\Omega$ ECs and zero in the others; (iii) maximizing the probability to reach good$_\Omega$ ECs is exactly how our algorithms operate.

▶ **Theorem 17.** *The threshold probability problem is*

(a) P*-complete for fixed window parity and fixed window mean-payoff objectives, and pure polynomial-memory optimal strategies can be constructed in polynomial time. Furthermore, polynomial memory is in general necessary.*

(b) P*-complete for bounded window parity objectives, and pure memoryless optimal strategies can be constructed in polynomial time;*

(c) *in* NP ∩ coNP *and as hard as mean-payoff games for bounded window mean-payoff objectives, and pure memoryless optimal strategies can be constructed in pseudo-polynomial time.*

The results follow from Thm. 15 and the MEC classification in the BW-good$_\mathsf{mp}$ case being the only non-polynomial operation of our algorithms. Plugging pure memoryless optimal strategies for reaching good$_\Omega$ MECs (granted by [4]) to our MEC strategies yields the upper bounds on memory. Hardness is essentially obtained through the two-player game interpretation of ECs [14, 17].

7 Limitations and perspectives

We summarized our results and compared them to the state of the art in Sect. 1. Here, we discuss the limitations of our work and some extensions within arm's reach.

Direct bounded window objectives. We left out a window objective considered in games [17, 14]: the *direct bounded* one, $\mathsf{DBW}_\Omega = \bigcup_{\lambda>0} \mathsf{DFW}_\Omega(\lambda)$. It is maybe not the most natural as it is *not* prefix-independent, yet allows to close the windows of a run in an arbitrarily large number of steps bounded along the run. This variant gives rise to complex behaviors in MDPs, notably due to its interaction with the almost-sure reachability of ECs.

Consider the MDP in Fig. 3 and objective DBW_mp. A window opens due to a. The only way to close it is to use b up to the point where the running sum becomes non-negative. When it does, all windows are closed and the controller may switch to s_5. Observe that

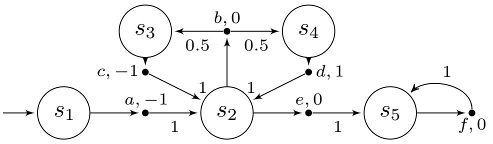


Figure 3 There exists a strategy σ ensuring $\mathsf{AS}^{\sigma}_{\mathcal{M},s_1}[\mathsf{DBW}_{\mathsf{mp}}]$ but it requires infinite memory as it needs to use b up to the point where the running sum becomes non-negative, then switch to e.

taking b repeatedly induces a *symmetric random walk* [29]. Classical probability results ensure that a non-negative sum will be obtained almost-surely, but the number of times b is played must remain unbounded (as for any bounded number, there exists a strictly positive probability to obtain only -1's for example). Thus, there exists an infinite-memory strategy σ such that $\mathsf{AS}^{\sigma}_{\mathcal{M},s_1}[\mathsf{DBW}_{\mathsf{mp}}]$, but no finite-memory one can do as good.

Now, let $\delta(s_2, b) = \{s_3 \mapsto 0.6, s_4 \mapsto 0.4\}$: the random walk becomes *asymmetric*, with a strictly positive chance to diverge toward $-\infty$. While the best possible strategy is still the one defined above, it only satisfies the objective with probability strictly less than one.

What do we observe? First, *infinite-memory strategies are required*, which is a problem for practical applications. Second, even for *qualitative* questions (is the probability zero or one?), the actual probabilities of the MDP must be considered, not only the existence of a transition. This is in stark contrast to most problems in MDPs [4]: in that sense, the direct bounded window objective is not well-behaved. This is due to the above connection with random walks. It is well-known that complex random walks are difficult to tackle for verification and synthesis: e.g., even simple asymmetric random walks are not *decisive* MCs, a large and robust class of MCs where reachability questions can be answered [1].

Markov chains. Our work focuses on the threshold probability problem for MDPs, and the corresponding strategy synthesis problem. Better complexities could possibly be obtained for MCs, where there are no non-deterministic choices. To achieve this, a natural direction would be to focus on the classification of ECs (Sect. 5), the bottleneck of our approach: for MCs, this classification would involve *one-player window games* (for the opponent), whose complexity has yet to be explored and would certainly be lower than for two-player games.

However, complexity is unlikely to be much lower: all parity variants are already in P, and the high complexity of $\mathsf{DFW}_{\mathsf{mp}}$ would remain: a construction similar to the PSPACE-hardness (Thm. 5) easily shows this problem to be PP-hard, already for *acyclic* MCs (again using [30]).

Expected value problem. Given an MDP $\mathcal{M}$ and an initial state s, we may be interested in synthesizing a strategy σ that minimizes the *expected window size* for a fixed window objective (say $\mathsf{FW}_{\Omega}(\lambda)$), which we straightforwardly define as $\mathbb{E}^{\sigma}_{\mathcal{M},s,\Omega}(\lambda) = \sum_{\lambda>0}^{\infty} \lambda \cdot \mathbb{P}^{\sigma}_{\mathcal{M},s}[\mathsf{FW}_{\Omega}(\lambda) \setminus \mathsf{FW}_{\Omega}(\lambda - 1)]$, with $\mathsf{FW}_{\Omega}(0) = \emptyset$. This meets the natural desire to build strategies that strive to maintain the *best time bounds possible in their local environment* (e.g., EC of $\mathcal{M}$). Note that this is totally different from the value function used in [8].

For prefix-independent variants, *we already have all the necessary machinery* to solve this problem. First, we refine the classification process to identify the best window size achievable in each MEC, if any. Indeed, if a MEC is λ-good, it necessarily is for some λ between one and the upper bound derived from the game-theoretic interpretation (Rmk. 11): we determine the smallest value of λ for each MEC via a binary search coupled with the classification procedure. Second, using classical techniques (e.g., [37]), we contract each MEC to a single-state EC, and give it a weight that represents the best window size we can ensure in it (hence this weight may be infinite if a MEC is not BW-good). Finally, we construct a global strategy that favors reaching MECs with the lowest weights, for example by synthesizing a strategy minimizing the classical mean-payoff value. Note that if λ-good MECs cannot be reached almost-surely, the expected value will be infinite, as wanted. Observe that *such an approach maintains tractability*, as we end up with a polynomial-time algorithm.

Direct variants require more involved techniques, as the unfoldings of Sect. 4 are strongly linked to the window size λ, and cannot be easily combined for different values of λ.

Multi-objective problems. Window games have been studied in the multidimension setting, where several weight (resp. priority) functions are given, and the objective is the intersection of all one-dimension objectives [17, 14]. Again, *our generic approach supports effortless extension to this setting.* In the direct case, unfoldings of Sect. 4 can be generalized to multiple dimensions, as in [17, 14]. For prefix-independent variants, the EC classification needs to be adapted to handle multidimension window games, which we can solve using [17, 14]. Then, we also need to consider a multi-objective reachability problem [37]. While almost all cases of multidimension window games are EXPTIME-complete, decidability of the bounded mean-payoff case is still open, but however known to be non-primitive recursive hard.

Tool support. Thanks to its low complexity and its adequacy w.r.t. applications, our window framework lends itself well to tool development. We are currently building a tool suite for MDPs with window objectives based on the main results of this paper along with the aforementioned extensions. Our aim is to provide a dedicated extension of Storm, a cutting-edge probabilistic model checker [23].

References

1 Parosh Aziz Abdulla, Noomene Ben Henda, and Richard Mayr. Decisive Markov Chains. *Logical Methods in Computer Science*, 3(4), 2007. doi:10.2168/LMCS-3(4:7)2007.

2 Pranav Ashok, Krishnendu Chatterjee, Przemyslaw Daca, Jan Kretínský, and Tobias Meggendorfer. Value Iteration for Long-Run Average Reward in Markov Decision Processes. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, volume 10426 of *Lecture Notes in Computer Science*, pages 201–221. Springer, 2017. doi:10.1007/978-3-319-63387-9_10.

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 Catriel Beeri. On the Membership Problem for Functional and Multivalued Dependencies in Relational Databases. *ACM Trans. Database Syst.*, 5(3):241–259, 1980. doi:10.1145/320613.320614.

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. *CoRR*, abs/1812.09298, 2018. arXiv:1812.09298.

9 Patricia Bouyer, Mauricio González, Nicolas Markey, and Mickael Randour. Multi-weighted Markov Decision Processes with Reachability Objectives. In Andrea Orlandini and Martin Zimmermann, editors, *Proceedings Ninth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2018, Saarbrücken, Germany, 26-28th September 2018.*, volume 277 of *EPTCS*, pages 250–264, 2018. doi:10.4204/EPTCS.277.18.

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. *CoRR*, abs/1901.03571, 2019. arXiv:1901.03571.

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 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.

19 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.

20 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`.

21 Wojciech Czerwinski, Laure Daviaud, Nathanaël Fijalkow, Marcin Jurdzinski, Ranko Lazic, and Pawel Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. *CoRR*, abs/1807.10546, 2018. `arXiv:1807.10546`.

22 Laure Daviaud, Marcin Jurdzinski, and Ranko Lazic. A pseudo-quasi-polynomial algorithm for mean-payoff parity games. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 325–334. ACM, 2018. `doi:10.1145/3209108.3209162`.

23 Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A Storm is Coming: A Modern Probabilistic Model Checker. In Rupak Majumdar and Viktor Kuncak, editors, *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part II*, volume 10427 of *Lecture Notes in Computer Science*, pages 592–600. Springer, 2017. `doi:10.1007/978-3-319-63390-9_31`.

24 John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is PSPACE-complete. *Inf. Comput.*, 243:26–36, 2015. `doi:10.1016/j.ic.2014.12.004`.

25 Nathanaël Fijalkow, Pawel Gawrychowski, and Pierre Ohlmann. The complexity of mean payoff games using universal graphs. *CoRR*, abs/1812.07072, 2018. `arXiv:1812.07072`.

26 Jerzy Filar and Koos Vrieze. *Competitive Markov decision processes.* Springer, 1997.

27 Thomas Gawlitza and Helmut Seidl. Games through Nested Fixpoints. In Ahmed Bouajjani and Oded Maler, editors, *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, volume 5643 of *Lecture Notes in Computer Science*, pages 291–305. Springer, 2009. `doi:10.1007/978-3-642-02658-4_24`.

28 Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science.* Springer, 2002. `doi:10.1007/3-540-36387-4`.

29 Charles M. Grinstead and J. Laurie Snell. *Introduction to probability.* American Mathematical Society, 1997.

30 Christoph Haase and Stefan Kiefer. The Odds of Staying on Budget. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 234–246. Springer, 2015. `doi:10.1007/978-3-662-47666-6_19`.

31 Arnd Hartmanns, Sebastian Junges, Joost-Pieter Katoen, and Tim Quatmann. Multi-cost Bounded Reachability in MDP. In Dirk Beyer and Marieke Huisman, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II*, volume 10806 of *Lecture Notes in Computer Science*, pages 320–339. Springer, 2018. `doi:10.1007/978-3-319-89963-3_19`.

32 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`.

33 Neil Immerman. Number of Quantifiers is Better Than Number of Tape Cells. *J. Comput. Syst. Sci.*, 22(3):384–406, 1981. `doi:10.1016/0022-0000(81)90039-8`.

34 Marcin Jurdzinski. Deciding the Winner in Parity Games is in UP ∩ co-UP. *Inf. Process. Lett.*, 68(3):119–124, 1998. `doi:10.1016/S0020-0190(98)00150-1`.

35 Mickael Randour. Automated Synthesis of Reliable and Efficient Systems Through Game Theory: A Case Study. In *Proc. of ECCS 2012*, Springer Proceedings in Complexity XVII, pages 731–738. Springer, 2013. `doi:10.1007/978-3-319-00395-5_90`.

36 Mickael Randour, Jean-François Raskin, and Ocan Sankur. 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.

37 Mickael Randour, Jean-François Raskin, and Ocan Sankur. Percentile queries in multi-dimensional Markov decision processes. *Formal Methods in System Design*, 50(2-3):207–248, 2017. doi:10.1007/s10703-016-0262-7.

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 fuer Informatik, 2018. doi:10.4230/LIPIcs.FSTTCS.2018.38.

39 Stephen D. Travers. The complexity of membership problems for circuits over sets of integers. *Theor. Comput. Sci.*, 369(1-3):211–229, 2006. doi:10.1016/j.tcs.2006.08.017.

40 Moshe Y. Vardi. Automatic Verification of Probabilistic Concurrent Finite-State Programs. In *Proc. of FOCS*, pages 327–338. IEEE, 1985.

Computing Probabilistic Bisimilarity Distances for Probabilistic Automata

Giorgio Bacci
Department of Computer Science, Aalborg University, DK
grbacci@cs.aau.dk

Giovanni Bacci
Department of Computer Science, Aalborg University, DK
giovbacci@cs.aau.dk

Kim G. Larsen
Department of Computer Science, Aalborg University, DK
kgl@cs.aau.dk

Radu Mardare
Department of Computer Science, Aalborg University, DK
mardare@cs.aau.dk

Qiyi Tang
Department of Computing, Imperial College, London, UK
qiyi.tang@imperial.ac.uk

Franck van Breugel
DisCoVeri Group, Dept. of Electrical Engineering and Computer Science, York University, Canada
franck@eecs.yorku.ca

Abstract

The probabilistic bisimilarity distance of Deng et al. has been proposed as a robust quantitative generalization of Segala and Lynch's probabilistic bisimilarity for probabilistic automata. In this paper, we present a novel characterization of the bisimilarity distance as the solution of a simple stochastic game. The characterization gives us an algorithm to compute the distances by applying Condon's simple policy iteration on these games. The correctness of Condon's approach, however, relies on the assumption that the games are *stopping*. Our games may be non-stopping in general, yet we are able to prove termination for this extended class of games. Already other algorithms have been proposed in the literature to compute these distances, with complexity in $\mathbf{UP} \cap \mathbf{coUP}$ and **PPAD**. Despite the theoretical relevance, these algorithms are inefficient in practice. To the best of our knowledge, our algorithm is the first practical solution. In the proofs of all the above-mentioned results, an alternative presentation of the Hausdorff distance due to Mémoli plays a central rôle.

2012 ACM Subject Classification Theory of computation → Program verification; Theory of computation → Models of computation; Mathematics of computing → Probability and statistics

Keywords and phrases Probabilistic automata, Behavioural metrics, Simple stochastic games, Simple policy iteration algorithm

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.9

Related Version A full version of the paper is available at `http://arxiv.org/abs/1907.01768`.

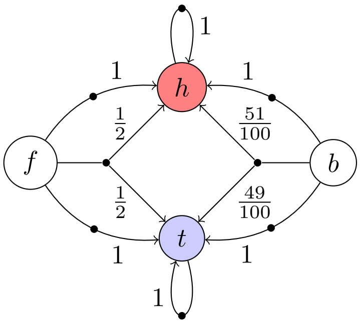
Funding Giovanni Bacci and Kim G. Larsen are funded by the ERC-Project LASSO, Radu Mardare is funded by the ASAP Project, Franck van Breugel is supported by Natural Sciences and Engineering Research Council of Canada.

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 9; pp. 9:1–9:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In [18], Giacalone et al. observed that for reasoning about the behaviour of probabilistic systems, rather than equivalences, a notion of distance is more reasonable in practice since it permits to capture the degree of difference between two states. This observation motivated the study of behavioural pseudometrics, that generalize behavioural equivalences in the sense that, when the distance is zero then the two states are behaviourally equivalent.

The systems we consider in this paper are labelled *probabilistic automata*. This model was introduced by Segala [32] to capture both nondeterminism (hence, concurrency) and probabilistic behaviours. The labels on states are used to express that certain properties of interest hold in particular states. Below we consider an example of probabilistic automaton describing two gamblers, f and b, deciding on which team to bet on in a football match.



Typically the two gamblers know the team to bet on, but occasionally they prefer to toss a coin to make a decision. This is represented by the three probabilistic transitions in the state f. The first two take f to state h (head) or t (tail) with probability one, the last takes f to states h and t with probability $\frac{1}{2}$ each. The difference between f and b is that the former uses a fair coin while the latter uses a biased coin landing on heads with slightly higher probability. Once the decision is taken, it is not changed anymore. This is seen on states h and t which have a single probabilistic transition taking the state to itself with probability one. The states h and t have distinct labels, here represented by colours.

A behavioural pseudometric for probabilistic automata capturing this difference is the *probabilistic bisimilarity distance* by Deng et al. [12], introduced as a robust generalization of Segala and Lynch's probabilistic bisimilarity [33]. The key ingredients of this pseudometric are the Hausdorff metric [19] and the Kantorovich metric [22], respectively used to capture nondeterministic and probabilistic behaviour. In the example above, the behaviours of the states h and t are very different since their labels are different. As a result, their probabilistic bisimilarity distance is one. On the other hand, the behaviours of the states f and b are very similar, which is reflected by the fact that their probabilistic bisimilarity distance is $\frac{1}{100}$.

The first attempt to compute the above distance is due to Chen et al. [9], who proposed a doubly exponential-time procedure to approximate the distances up to any degree of accuracy. The complexity was later improved to **PSPACE** by Chattarjee et al. [6, 7]. Their solutions exploit the decision procedure for the existential fragment of the first-order theory of reals. It is worth noting that [9, 6] consider the pseudometric that does not discount the future (a.k.a. *undiscounted* distance) which entails additional algorithmic challenges. Later, Fu [16] showed that the distances have rational values and that computing the discounted distance can be done in polynomial time by using a value-iteration procedure in combination with the continued fraction algorithm [31, Section 6]. As for the undiscounted distance, he showed that the threshold problem, i.e., deciding whether the distance is smaller than a given rational, is in $\mathbf{NP} \cap \mathbf{coNP}$[1]. Van Breugel and Worrell [40] have later shown that the problem is in **PPAD**, which is short for polynomial parity argument in a directed graph. Notably, their proof exploits a characterization of the distance as a simple stochastic game. Despite the theoretical relevance of the above algorithms, their implementations are inefficient in practice since they require the use of exact computer arithmetic making it difficult to handle models with more than five states. In this paper, we propose an alternative approach that is inspired by the successful implementations of similar pseudometrics on Markov chains [1, 37, 38].

Our solution is based on a novel characterization of the probabilistic bisimilarity distance as the solution of a simple stochastic game. Stochastic games were introduced by Shapley [34]. A simplified version of these games, called *simple stochastic games*, were studied by Condon [11]. Several algorithms have been proposed to compute the value function of a simple stochastic game, many using policy iteration. Condon [10] proposed an algorithm, known as *simple policy iteration*, that switches only one non-optimal choice per iteration. The correctness of Condon's algorithm, however, relies on the assumption that the game is *stopping*.

It turns out that the simple stochastic games characterizing the probabilistic bisimilarity distances are stopping only when the distances discount the future. In the case the distance is non-discounting, the corresponding games may not be stopping. To recover correctness of the policy iteration procedure we adapt Condon's simple policy iteration algorithm by adding a non-local update of the strategy of the min player and an extra termination condition based on a notion of "self-closed" relation due to Fu [16]. The practical efficiency of our algorithm has been evaluated on a significant set of randomly generated probabilistic automata. The results show that our algorithm performs better than the corresponding iterative algorithms proposed for the discounted distances in [16], even though the theoretical complexity of our proposal is exponential in the worst case (cf. [37]) whereas Fu's is polynomial.

The implementation of the algorithms exploits a coupling structure characterization of the distance that allows us to skip the construction of the simple stochastic game which may result in an exponential blow up of the memory required for storing the game. Finally, we remark that in the proofs of most of the above mentioned results a dual representation of the Hausdorff distance due to Mémoli [26] plays a central rôle.

2 Preliminaries and Notation

The set of functions f from X to Y is denoted by Y^X. We denote by $f[x/y] \in Y^X$ the *update of f* at $x \in X$ with $y \in Y$, defined by $f[x/y](x') = y$ if $x' = x$, otherwise $f[x/y](x') = f(x')$.

A (1-bounded) *pseudometric* on a set X is a function $d \colon X \times X \to [0,1]$ such that, $d(x,x) = 0$, $d(x,y) = d(y,x)$, and $d(x,y) \leq d(x,z) + d(z,y)$ for all $x, y, z \in X$.

Kantorovich lifting. A (discrete) probability distribution on X is a function $\mu \colon X \to [0,1]$ such that $\sum_{x \in X} \mu(x) = 1$, and its support is $supp(\mu) = \{x \in X \mid \mu(x) > 0\}$. We denote by $\mathcal{D}(X)$ the set of probability distributions on X. A pseudometric d on X can be lifted to a pseudometric on probability distributions in $\mathcal{D}(X)$ by means of the Kantorovich lifting [41].

The *Kantorovich lifting* of $d \in [0,1]^{X \times X}$ on distributions $\mu, \nu \in \mathcal{D}(X)$ is defined by

$$\mathcal{K}(d)(\mu,\nu) = \min\left\{ \textstyle\sum_{x,y \in X} d(x,y) \cdot \omega(x,y) \mid \omega \in \Omega(\mu,\nu) \right\}, \qquad \text{(Kantorovich lifting)}$$

[1] The same proof can be adapted to show that the decision problem is in $\mathbf{UP} \cap \mathbf{coUP}$ [17]. Recall that **UP** contains those problems in **NP** with a unique accepting computation.

where $\Omega(\mu, \nu)$ denotes the set of *measure-couplings* for the pair (μ, ν), i.e., distributions $\omega \in \mathcal{D}(X \times X)$ such that, for all $x \in X$, $\sum_{y \in X} \omega(x, y) = \mu(x)$ and $\sum_{y \in X} \omega(y, x) = \nu(x)$. It is a well known fact that the Kantorovich lifting can be equivalently defined by ranging ω over the set of vertices $V(\Omega(\mu, \nu))$ of the polytope $\Omega(\mu, \nu)$. Thus, a minimum is always attained at a vertex. Furthermore, if the set X is finite, the set $V(\Omega(\mu, \nu))$ is finite too.

Hausdorff lifting. A pseudometric d on X can be lifted to nonempty subsets of X by means of the Hausdorff lifting. The *Hausdorff lifting* of $d \in [0, 1]^{X \times X}$ on nonempty subsets $A, B \subseteq X$ is defined by

$$\mathcal{H}(d)(A, B) = \max \{\sup_{a \in A} \inf_{b \in B} d(a, b), \sup_{b \in B} \inf_{a \in A} d(a, b)\} \ . \quad \text{(HAUSDORFF LIFTING)}$$

Following Mémoli [26, Lemma 3.1], the Hausdorff lifting has a dual characterization in terms of *set-couplings*[2]. Given $A, B \subseteq X$, a set-coupling for (A, B) is a relation $R \subseteq X \times X$ with left and right projections respectively equal to A and B, i.e., $\{a \mid \exists b \in X. \, a \, R \, b\} = A$ and $\{b \mid \exists a \in X. \, a \, R \, b\} = B$. We write $\mathcal{R}(A, B)$ for the set of the set-couplings for (A, B).

▶ **Theorem 1.** $\mathcal{H}(d)(A, B) = \inf\{\sup_{(a,b) \in R} d(a, b) \mid R \in \mathcal{R}(A, B)\}$.

Clearly, for A, B finite, inf and sup in Theorem 1 can be replaced by min and max, respectively.

3 Probabilistic Automata and Probabilistic Bisimilarity Distance

In this section we recall the definitions of *probabilistic automaton*, Segala and Lynch's *probabilistic bisimulation* [33], and *probabilistic bisimilarity distance* of Deng et al. [12].

▶ **Definition 2.** *A* probabilistic automaton *(PA) is a tuple $\mathcal{A} = (S, L, \rightarrow, \ell)$ consisting of a nonempty finite set S of states, a finite set of labels L, a finite total transition relation $\rightarrow \subseteq S \times \mathcal{D}(S)$, and a labelling function $\ell \colon S \rightarrow L$.*

Note that for simplicity we assume the transition relation $\rightarrow$ to be total, that is, for all $s \in S$, there exists a $\mu \in \mathcal{D}(S)$ such that $(s, \mu) \in \rightarrow$. For the remainder of this paper we fix a probabilistic automaton $\mathcal{A} = (S, L, \rightarrow, \ell)$. We write $s \rightarrow \mu$ to denote $(s, \mu) \in \rightarrow$ and use $\delta(s)$ to denote the set $\{\mu \mid s \rightarrow \mu\}$ of successor distributions of s.

Next we recall the notion of probabilistic bisimilarity due to Segala and Lynch [33] for probabilistic automata. Their definition exploits the notion of lifting of a relation $R \subseteq S \times S$ on states to a relation $\tilde{R} \subseteq \mathcal{D}(S) \times \mathcal{D}(S)$ on probability distributions on states, originally introduced by Jonsson and Larsen [20], and defined by $\mu \, \tilde{R} \, \nu$ if there exists a measure-coupling $\omega \in \Omega(\mu, \nu)$ such that *supp*$(\omega) \subseteq R$.

▶ **Definition 3.** *A relation $R \subseteq S \times S$ is a* probabilistic bisimulation *if whenever $s \, R \, t$,*

- *$\ell(s) = \ell(t)$,*
- *if $s \rightarrow \mu$ then there exists $t \rightarrow \nu$ such that $\mu \, \tilde{R} \, \nu$, and*
- *if $t \rightarrow \nu$ then there exists $s \rightarrow \mu$ such that $\mu \, \tilde{R} \, \nu$.*

Two states $s, t \in S$ are probabilistic bisimilar*, written $s \sim t$, if they are related by some probabilistic bisimulation.*

[2] Mémoli uses the terminology "correspondence." To avoid confusion, we adopted the same terminology used in [29, Section 10.6].

Intuitively, two states are probabilistic bisimilar if they have the same label and each transition of the one state to a distribution μ can be matched by a transition of the other state to a distribution ν assigning the same probability to states that behave the same, and vice versa. Probabilistic bisimilarity is an equivalence relation and the largest probabilistic bisimulation.

Deng et al. [12] proposed a family of 1-bounded pseudometrics $\mathbf{d}_\lambda$, parametric on a *discount factor* $\lambda \in (0, 1]$, called *probabilistic bisimilarity pseudometrics*. The pseudometrics $\mathbf{d}_\lambda$ are defined as the least fixed-point of the functions $\Delta_\lambda \colon [0,1]^{S\times S} \to [0,1]^{S\times S}$

$$\Delta_\lambda(d)(s,t) = \begin{cases} 1 & \text{if } \ell(s) \neq \ell(t) \\ \lambda \cdot \mathcal{H}(\mathcal{K}(d))(\delta(s), \delta(t)) & \text{otherwise}\,. \end{cases}$$

The well-definition of $\mathbf{d}_\lambda$ follows by Knaster-Tarski's fixed point theorem, given the fact that Δ_λ is a monotone function on the complete partial order of $[0, 1]$-valued functions on $S \times S$ ordered point-wise by $d \sqsubseteq d'$ iff for all $s, t \in S$, $d(s, t) \leq d'(s, t)$.

The fact that probabilistic bisimilarity distances provide a quantitative generalization of bisimilarity is captured by the following theorem due to Deng et al. [12, Corollary 2.14].

▶ **Theorem 4.** *For all $\lambda \in (0, 1]$, $\mathbf{d}_\lambda(s, t) = 0$ if and only if $s \sim t$.*

4 Probabilistic Bisimilarity Distance as a Simple Stochastic Game

A *simple stochastic game* (SSG) consists of a finite directed graph whose vertices are partitioned into sets of *0-sinks*, *1-sinks*, *max vertices*, *min vertices*, and *random vertices*. The game is played by two players, the *max player* and the *min player*, with a single token. At each step of the game, the token is moved from a vertex to one of its successors. At a min vertex the min player chooses the successor, at a max vertex the max player chooses the successor, and at a random vertex the successor is chosen randomly according to a prescribed probability distribution. The max player wins a play of the game if the token reaches a 1-sink and the min player wins if the play reaches a 0-sink or continues forever without reaching a sink. Since the game is stochastic, the max player tries to maximize the probability of reaching a 1-sink whereas the min player tries to minimize that probability.

▶ **Definition 5.** *A* simple stochastic game *is a tuple (V, E, P) consisting of*

- *a finite directed graph (V, E) such that*
  - *V is partitioned into the sets: V_0 of* 0-sinks*, V_1 of* 1-sinks*, $V_{\max}$ of* max vertices*, $V_{\min}$ of* min vertices*, and V_{rnd} of* random vertices*;*
  - *the vertices in V_0 and V_1 have outdegree zero and all other vertices have outdegree at least one, and*
- *a function $P \colon V_{\text{rnd}} \to \mathcal{D}(V)$ such that for all $v \in V_{\text{rnd}}$ and $w \in V$, $P(v)(w) > 0$ iff $(v, w) \in E$.*

A *strategy*, also known as *policy*, for the min player is a function $\sigma_{\min} \colon V_{\min} \to V$ that assigns the target of an outgoing edge to each min vertex, that is, for all $v \in V_{\min}$, $(v, \sigma_{\min}(v)) \in E$. Likewise, a strategy for the max player is a function $\sigma_{\max} \colon V_{\max} \to V$ that assigns the target of an outgoing edge to each max vertex. These strategies are known as *pure stationary* strategies. We can restrict ourselves to these strategies since both players of a simple stochastic game have optimal strategies of this type (see, for example, [25]).

Such strategies determine a sub-game in which each max vertex and each min vertex has outdegree one (see [11, Section 2] for details). Such a game can naturally be viewed as a Markov chain. We write $\phi_{\sigma_{\min},\sigma_{\max}} \colon V \to [0, 1]$ for the function that gives the probability of a vertex in this Markov chain to reach a 1-sink.

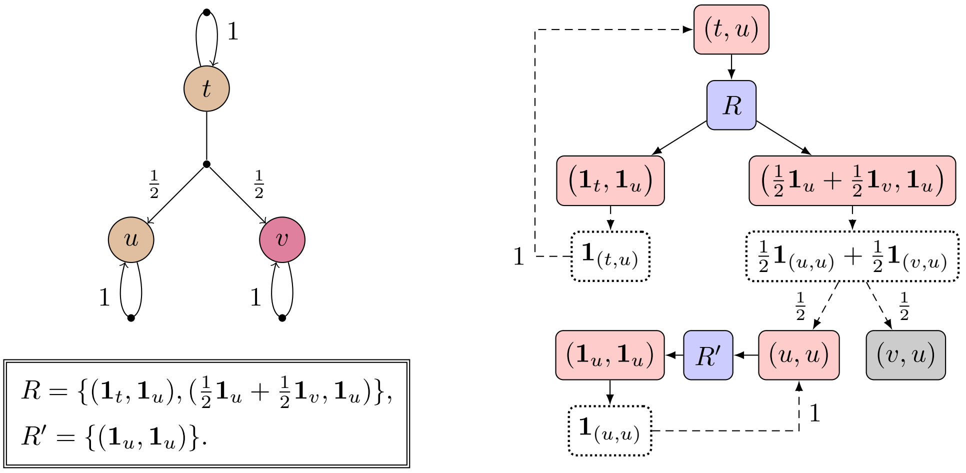

t
u
v
1
$\frac{1}{2}$
$\frac{1}{2}$
1
1
$R = \{(\mathbf{1}_t, \mathbf{1}_u), (\frac{1}{2}\mathbf{1}_u + \frac{1}{2}\mathbf{1}_v, \mathbf{1}_u)\}$,
$R' = \{(\mathbf{1}_u, \mathbf{1}_u)\}$.
(t, u)
R
$(\mathbf{1}_t, \mathbf{1}_u)$
$(\frac{1}{2}\mathbf{1}_u + \frac{1}{2}\mathbf{1}_v, \mathbf{1}_u)$
1
$\mathbf{1}_{(t,u)}$
$\frac{1}{2}\mathbf{1}_{(u,u)} + \frac{1}{2}\mathbf{1}_{(v,u)}$
$\frac{1}{2}$
$\frac{1}{2}$
$(\mathbf{1}_u, \mathbf{1}_u)$
R'
(u, u)
(v, u)
$\mathbf{1}_{(u,u)}$
1

Figure 1 (Top left:) A probabilistic automaton and (Right:) the associated simple stochastic game constructed as in Definition 8 for $\lambda = 1$ (only the portion reachable from (t, u) is shown), where $\mathbf{1}_x$ denotes the Dirac distribution concentrated at x.

The *value function* $\phi\colon V \to [0, 1]$ of a SSG is defined as $\min_{\sigma_{\min}} \max_{\sigma_{\max}} \phi_{\sigma_{\min},\sigma_{\max}}$. It is folklore that the value function of a simple stochastic game can be characterised as the least fixed point of the following monotone function (see, for example, [21, Section 2.2 and 2.3]).

▶ **Definition 6.** *The function* $\Phi\colon [0, 1]^V \to [0, 1]^V$ *is defined by*

$$\Phi(f)(v) = \begin{cases} 0 & \text{if } v \in V_0 \\ 1 & \text{if } v \in V_1 \\ \max_{(v,w)\in E} f(w) & \text{if } v \in V_{\max} \\ \min_{(v,w)\in E} f(w) & \text{if } v \in V_{\min} \\ \sum_{(v,w)\in E} P(v)(w)\, f(w) & \text{if } v \in V_{\text{rnd}} \end{cases}$$

▶ **Proposition 7.** *The function* Φ *is monotone and nonexpansive.*

A Probabilistic Bisimilarity Game. Fix a probabilistic automaton $\mathcal{A}$ and $\lambda \in (0, 1]$. We will characterise the probabilistic bisimilarity distance as values of a simple stochastic game, which we call the *probabilistic bisimilarity game*, where the min player tries to show that two states are probabilistic bisimilar, while the max player tries to prove the opposite.

In our probabilistic bisimilarity game, there is a vertex (s, t) for each pair states s and t in $\mathcal{A}$. If $\ell(s) \neq \ell(t)$ then the vertex (s, t) is a 1-sink. Otherwise, (s, t) is a min vertex. In this vertex, the min player selects a set $R \in \mathcal{R}(\delta(s), \delta(t))$ of pairs of transitions. This set R captures potential matchings of transitions from state s and state t. Subsequently, the max player chooses a pair of transitions from the set R. Once the max player has chosen a pair (μ, ν) from the set R corresponding to the transitions $s \to \mu$ and $t \to \nu$, the min player can choose a measure-coupling $\omega \in \Omega(\mu, \nu)$. To ensure that the game graph is finite, we restrict our attention to the vertices $V(\Omega(\mu, \nu))$ of the polytope $\Omega(\mu, \nu)$. Such a measure-coupling ω captures a matching of the probability distributions μ and ν. Recall that a coupling is a probability distribution on $S \times S$. From a random vertex ω, the game proceeds to vertex (u, v) with probability $\lambda \cdot \omega(u, v)$ and to the 0-sink vertex $\perp$ with probability $1 - \lambda$. Intuitively, the choices of $R \in \mathcal{R}(\delta(s), \delta(t))$ and then $(\mu, \nu) \in R$, performed respectively by the min and the max player, correspond to the min and max of Theorem 1; analogously, the selection of $\omega \in V(\Omega(\mu, \nu))$ by the min player models the min in the definition of the Kantorovich lifting.

Formally, our probabilistic bisimilarity game for the automaton $\mathcal{A}$ is defined as follows.

▶ **Definition 8.** *Let $\lambda \in (0, 1]$. The* probabilistic bisimilarity game (V, E, P) *is defined by*

- $V_0 = \{\bot\}$,
- $V_1 = \big\{(s,t) \in S \times S \mid \ell(s) \neq \ell(t)\big\}$,
- $V_{\max} = \bigcup \big\{\mathcal{R}(\delta(s), \delta(t)) \mid (s,t) \in V_{\min}\big\}$,
- $V_{\min} = \big\{(s,t) \in S \times S \mid \ell(s) = \ell(t)\big\} \cup \bigcup \big\{R \mid R \in V_{\max}\big\}$,
- $V_{\mathrm{rnd}} = \bigcup \big\{V(\Omega(\mu,\nu)) \mid (\mu,\nu) \in V_{\min}\big\}$,

$$\begin{aligned}E = {} & \big\{((s,t), R) \mid (s,t) \in V_{\min} \wedge R \in \mathcal{R}(\delta(s), \delta(t))\big\} \cup \\ & \big\{(R, (\mu,\nu)) \mid R \in V_{\max} \wedge (\mu,\nu) \in R\big\} \cup \\ & \big\{((\mu,\nu), \omega) \mid (\mu,\nu) \in V_{\min} \wedge \omega \in V(\Omega(\mu,\nu))\big\} \cup \\ & \big\{(\omega, (u,v)) \mid \omega \in V_{\mathrm{rnd}} \wedge (u,v) \in \mathit{supp}(\omega)\big\} \cup \big\{(\omega, \bot) \mid \omega \in V_{\mathrm{rnd}}\big\}\,,\end{aligned}$$

and, for all $\omega \in V_{\mathrm{rnd}}$ and $(s,t) \in \mathit{supp}(\omega)$, $P(\omega)((s,t)) = \lambda \cdot \omega(s,t)$ and $P(\omega)(\bot) = 1 - \lambda$.

By construction of the probabilistic bisimilarity game, there is a direct correspondence between the function Φ from Definition 6 associated to the probabilistic bisimilarity game and the function Δ_λ from Section 3 associated to the probabilistic automaton. From this correspondence it is straightforward that the respective least fixed points of Φ and Δ_λ agree, that is, the probabilistic bisimilarity distances of a probabilistic automaton are the values of the corresponding vertices of the probabilistic bisimilarity game.

▶ **Theorem 9.** *For all $s, t \in S$, $\mathbf{d}_\lambda(s,t) = \phi(s,t)$.*

The proof is similar to that of [40, Theorem 14].

Consider a state pair (s,t) with $s \sim t$. By Theorem 4, $\mathbf{d}_\lambda(s,t) = 0$. Hence, from Theorem 9 we can conclude that $\phi(s,t) = 0$. Therefore, by pre-computing probabilistic bisimilarity, (s,t) can be represented as a 0-sink, rather than a min vertex. For example, in Figure 1 this amounts to turn (u,u) into a 0-sink and disconnect it from its successors.

Games similar to the above introduced probabilistic bisimilarity game have been presented in [14, 40, 15, 24]. The game presented by van Breugel and Worrell in [40] is most closely related to our game. They also consider probabilistic automata and map a probabilistic automaton to a simple stochastic game. The only difference is that they use the original definition of the Hausdorff distance, whereas we use Mémoli's alternative characterization. The games described in [14, 15, 24] are not stochastic. Desharnais, Laviolette and Tracol [14] define an ϵ-probabilistic bisimulation game for probabilistic automata, where $\epsilon > 0$ captures the maximal amount of difference in behaviour that is allowed. Their measure of difference in behaviour is incomparable to our probabilistic bisimilarity distances (see [14, Section 6]). König and Mika-Michalski [24] generalize the game of Desharnais et al. in a categorical setting so that it is applicable to a large class of systems including probabilistic automata. Fijalkow, Klin and Panangaden [15] consider a more restricted class of systems, namely systems with probabilities but without nondeterminism. In the games in [14, 15, 24] players choose sets of states, a phenomenon that one does not encounter in our game.

5 A Coupling Characterisation of the Bisimilarity Distance

In this section we provide an alternative characterisation for the probabilistic bisimilarity distance $\mathbf{d}_\lambda$ based on the notion of coupling structure for a probabilistic automaton. This characterisation generalises the one by Chen et al. [8, Theorem 8] (see also [1, Theorem 8]) for the bisimilarity pseudometric of Desharnais et al. [13] for Markov chains. Our construction exploits Mémoli's dual characterisation of the Hausdorff distance (Theorem 1).

▶ **Definition 10.** *A* coupling structure *for $\mathcal{A}$ is a tuple $\mathcal{C} = (f, \rho)$ consisting of*

- *a map $f \colon \mathcal{D}(S) \times \mathcal{D}(S) \to \mathcal{D}(S \times S)$ such that, for all $\mu, \nu \in \mathcal{D}(S)$, $f(\mu,\nu) \in \Omega(\mu,\nu)$, and*
- *a map $\rho \colon S \times S \to 2^{\mathcal{D}(S)\times\mathcal{D}(S)}$, such that for all $s,t \in S$, $\rho(s,t) \in \mathcal{R}(\delta(s), \delta(t))$.*

For convenience, the components f and ρ of a coupling structure will be respectively called *measure-coupling map* and *set-coupling map*.

A coupling structure $\mathcal{C} = (f, \rho)$ induces a probabilistic automaton $\mathcal{A}_\mathcal{C}$ on $S \times S$ with transition relation $\to_\mathcal{C} \subseteq (S \times S) \times \mathcal{D}(S \times S)$, defined as $(s,t) \to_\mathcal{C} f(\mu,\nu)$ if $(\mu,\nu) \in \rho(s,t)$.

Let $\lambda \in (0,1]$. For each $\mathcal{C}$ we define the function $\Gamma^\mathcal{C}_\lambda \colon [0,1]^{S\times S} \to [0,1]^{S\times S}$ as

$$\Gamma^\mathcal{C}_\lambda(d)(s,t) = \begin{cases} 1 & \text{if } \ell(s) \neq \ell(t) \\ \lambda \cdot \max\{\sum_{u,v\in S} d(u,v)\cdot\omega(u,v) \mid (s,t) \to_\mathcal{C} \omega\} & \text{otherwise.} \end{cases}$$

One can easily verify that $\Gamma^\mathcal{C}_\lambda$ is well-defined and monotonic. Thus, by Knaster-Tarski's fixed point theorem, $\Gamma^\mathcal{C}_\lambda$ has a least fixed point, denoted by $\gamma^\mathcal{C}_\lambda$. As in [1], we call $\gamma^\mathcal{C}_\lambda$ the *λ-discounted discrepancy* w.r.t. $\mathcal{C}$ or simply *λ-discrepancy*.

▶ Remark 11. Note that, $\gamma^\mathcal{C}_1(s,t)$ is the maximal probability of reaching a pair of states (u,v) in the probabilistic automaton $\mathcal{A}_\mathcal{C}$ such that $\ell(u) \neq \ell(v)$ by starting from the state pair (s,t). It is well known that the maximal reachability probability can be computed in polynomial-time as the optimal solution of a linear program (see [5, Chapter 10] or [30, Chapter 6]). The linear program can be trivially generalized to compute $\gamma^\mathcal{C}_\lambda$, for any $\lambda \in (0,1]$.

▶ **Lemma 12.** *Let $\lambda \in (0,1]$. Then, $\Delta_\lambda(\gamma^\mathcal{C}_\lambda) \sqsubseteq \gamma^\mathcal{C}_\lambda$ for all coupling structures $\mathcal{C}$ of $\mathcal{A}$.*

The next lemma shows that the probabilistic bisimilarity distance can be characterised as the λ-discrepancy for a *vertex coupling structure*, that is, a coupling structure $\mathcal{C} = (f,\rho)$ such that $f(\mu,\nu) \in V(\Omega(\mu,\nu))$ for all $\mu,\nu \in \mathcal{D}(S)$.

▶ **Lemma 13.** *Let $\lambda \in (0,1]$. Then, $\mathbf{d}_\lambda = \gamma^\mathcal{C}_\lambda$ for some vertex coupling structure $\mathcal{C}$.*

▶ **Theorem 14.** *Let $\lambda \in (0,1]$. Then, the following hold:*

1. *$\mathbf{d}_\lambda = \sqcap\{\gamma^\mathcal{C}_\lambda \mid \mathcal{C}$ coupling structure for $\mathcal{A}\}$;*
2. *$s \sim t$ iff $\gamma^\mathcal{C}_\lambda(s,t) = 0$ for some vertex coupling structure $\mathcal{C}$ for $\mathcal{A}$.*

Proof. (1) follows by Knaster-Tarski's fixed point theorem, Lemmas 12, and 13; (2) follows by Theorem 4 and Lemma 13. ◀

Note that together with Lemma 13, Theorem 14.1 states that $\mathbf{d}_\lambda$ can obtained as the minimal λ-discrepancy obtained by ranging over the subset of vertex coupling structures.

▶ Remark 15 (On the relation with probabilistic bisimilarity games). The coupling structure characterization of the distance is strongly related to the simple stochastic game characterization presented in Section 4. Indeed, the notion of vertex coupling structure captures essentially the strategies for the min player on a probabilistic bisimilarity game in the following sense: the measure-coupling map component describes the strategy on the vertices of the form $(\mu,\nu) \in R$ for some $R \in V_{\max}$, while the set-coupling map deals with the description of the strategy on the min vertices $(s,t) \in S \times S$. The discrepancy $\gamma^\mathcal{C}_1$ captures the value w.r.t an optimal strategy for the max player when the min player has fixed their strategy a priori.

6 Computing the Bisimilarity Distance

We describe a procedure for computing the bisimilarity distances based on Condon's simple policy iteration algorithm [10]. Our procedure extends a similar one proposed in [37, 1] for computing the bisimilarity distance of Desharnais et al. [13] for Markov chains. The extension takes into account the additional presence of nondeterminism in the choice of the transitions.

Condon's simple policy iteration algorithm computes the values of a simple stochastic game provided that the game is *stopping*, i.e., for each pair of strategies for the min and max players the token reaches a 0-sink or 1-sink vertex with probability one.

As we have shown in Theorem 9, the probabilistic bisimilarity distances are the values of the corresponding vertices in the simple stochastic game given in Definition 8. Thus, if we prove that the game is stopping we can apply Condon's simple policy iteration algorithm to compute the probabilistic bisimilarity distances.

▶ **Proposition 16.** *For $\lambda \in (0,1)$, the simple stochastic game in Definition 8 is stopping.*

However, for $\lambda = 1$ the game in Definition 8 may not be stopping as shown below.

▶ **Example 17.** Consider the probabilistic automaton in Figure 1 and its associated probabilistic bisimilarity game. By choosing a strategy $\sigma_{\max}$ for the max player such that $\sigma_{\max}(R) = (\mathbf{1}_t, \mathbf{1}_u)$, the vertex (t,u) has probability zero to reach a sink. This can be seen in Figure 1, since there are no paths using the edge $(R, (\mathbf{1}_t, \mathbf{1}_u))$ leading to a sink.

In [37], by imposing the bisimilar state pairs to be 0-sinks, for the case of Markov chains the simple stochastic game was proven to be stopping. This method does not generalize to probabilistic automata. Indeed, Example 17 provides a counterexample even when bisimilar state pairs are 0-sinks.

In the remainder of the section, we provide a general algorithm to compute the bisimilarity distance for every $\lambda \in (0,1]$, by adapting Condon's simple policy iteration algorithm. Our solution will exploit the coupling characterization of the distance discussed in Section 5. This allows us to skip the construction of the simple stochastic game which may have size exponential in the number of states of the automaton.

6.1 Simple Policy Iteration Strategy

Condon's algorithm iteratively updates the strategies of the min and max players in turn, on the basis of the current over-approximation of the value of the game. Next we show how Condon's policy updates can be performed directly on coupling structures.

For the update of the coupling structure, will use measure-coupling maps of the form $k(d)(\mu,\nu) \in V(\Omega(\mu,\nu))$ and a set-coupling $h(d)(s,t) \in \mathcal{R}(\delta(s),\delta(t))$ such that

$$k(d)(\mu,\nu) \in \operatorname{argmin}\left\{ \textstyle\sum_{u,v\in S} \omega(u,v)\cdot d(u,v) \mid \omega \in V(\Omega(\mu,\nu)) \right\}, \text{ and} \tag{1}$$

$$h(d)(s,t) \in \operatorname{argmin}\left\{ \max_{(\mu,\nu)\in R} \mathcal{K}(d)(\mu,\nu) \mid R \in \mathcal{R}(\delta(s),\delta(t)) \right\}. \tag{2}$$

for $d \colon S \times S \to [0,1]$, $\mu,\nu \in \mathcal{D}(S)$, and $s,t \in S$.

The following lemma explains how the above ingredients can be used by the min player to improve their strategy.

▶ **Lemma 18.** *Let $\mathcal{C} = (f,\rho)$. If there exist $s,t \in S$ such that $\Delta_\lambda(\gamma_\lambda^{\mathcal{C}})(s,t) < \gamma_\lambda^{\mathcal{C}}(s,t)$ then, $\gamma_\lambda^{\mathcal{D}} \sqsubset \gamma_\lambda^{\mathcal{C}}$ for a coupling structure $\mathcal{D} = (k(\gamma_\lambda^{\mathcal{C}}), \rho[(s,t)/R])$, where $R = h(\gamma_\lambda^{\mathcal{C}})(s,t)$.*

Algorithm 1: Simple policy iteration algorithm computing $\mathbf{d}_\lambda$ for $\lambda \in (0,1)$.

```
1 Initialise $\mathcal{C} = (f, \rho)$ as an arbitrary vertex coupling structure for $\mathcal{A}$
2 while $\exists (s,t).\, \Delta_\lambda(\gamma_\lambda^{\mathcal{C}})(s,t) < \gamma_\lambda^{\mathcal{C}}(s,t)$ do
3     $R \leftarrow h(\gamma_\lambda^{\mathcal{C}})(s,t)$
4     $\mathcal{C} \leftarrow \big(k(\gamma_\lambda^{\mathcal{C}}),\, \rho[(s,t)/R]\big)$          /* update coupling structure */
5 end
6 return $\gamma_\lambda^{\mathcal{C}}$                                   /* $\gamma_\lambda^{\mathcal{C}} = \mathbf{d}_\lambda$ */
```

Lemma 18 suggests that $\mathcal{C} = (f, \rho)$ can be improved by replacing the measure-coupling map f with $k(\gamma_\lambda^{\mathcal{C}})$ and updating the set-coupling map ρ at (s,t) with $R = h(\gamma_\lambda^{\mathcal{C}})(s,t)$.

Note that a measure-coupling $k(d)(\mu,\nu)$ satisfying (1) can be computed by solving a linear program and ensuring that the optimal solution is a vertex of the polytope $\Omega(\mu,\nu)$ [28, 23]. A set-coupling $h(d)(s,t)$ satisfying (2) is the following:

$$R = \{(\mu, \phi(\mu)) \mid \mu \in \delta(s)\} \cup \{(\psi(\nu), \nu) \mid \nu \in \delta(t)\} \in \mathcal{R}(\delta(s), \delta(t)), \tag{3}$$

where ϕ, ψ are such that $\phi(\mu) \in \operatorname{argmin}_{\nu \in \delta(t)} \mathcal{K}(d)(\mu,\nu)$ and $\psi(\nu) \in \operatorname{argmin}_{\mu \in \delta(s)} \mathcal{K}(d)(\mu,\nu)$. The following lemma justifies our choice of $h(d)(s,t)$.

▶ **Lemma 19.** *Let R be as in* (3)*. Then $\mathcal{H}(\mathcal{K}(d))(\delta(s),\delta(t)) = \max_{(\mu,\nu)\in R} \mathcal{K}(d)(\mu,\nu)$.*

▶ Remark 20. The update procedure entailed by Lemma 18 can be performed in polynomial-time in the size of the probabilistic automaton $\mathcal{A}$. Indeed, $k(d)(\mu,\nu)$ can be obtained by solving a transportation problem in polynomial time [28, 23]. As for $h(d)(s,t)$, one can obtain $\phi(\mu)$ (resp. $\psi(\nu)$) by computing $\mathcal{K}(d)(\mu,\nu)$ in polynomial time and selecting the ν (resp. μ) ranging over $\delta(t)$ (resp. $\delta(s)$) that achieves the minimum.

Discounted case. The simple policy iteration algorithm for computing $\mathbf{d}_\lambda$ in the case $\lambda < 1$ is presented in Algorithm 1. The procedure starts by computing an initial vertex coupling structure $\mathcal{C}_0$ (line 1), e.g., by using the North-West corner method in polynomial time (see [35, pg. 180]). Then it continues by iteratively generating a sequence $\mathcal{C}_0, \mathcal{C}_1, \dots, \mathcal{C}_n$ of vertex coupling structures where $\mathbf{d}_\lambda = \gamma_\lambda^{\mathcal{C}_n}$. At each iteration, the current coupling structure $\mathcal{C}_i$ is tested for optimality (line 2) by checking whether the corresponding λ-discrepancy $\gamma_\lambda^{\mathcal{C}_i}$ is a fixed point for Δ_λ. If there exists $(s,t) \in S$ violating the equality $\gamma_\lambda^{\mathcal{C}_i} = \Delta_\lambda(\gamma_\lambda^{\mathcal{C}_i})$, it constructs $\mathcal{C}_{i+1}$ by updating $\mathcal{C}_i$ at (s,t) as prescribed by Lemma 18 (line 4). This guarantees that $\gamma_\lambda^{\mathcal{C}_i} \sqsupset \gamma_\lambda^{\mathcal{C}_{i+1}}$, i.e., a strict improvement of the λ-discrepancy towards the minimal one.

Termination follows by the fact that there are only finitely many vertex coupling structures for $\mathcal{A}$. Furthermore, the correctness of the output of the algorithm is due to the fact that, Δ_λ has a unique fixed point when $0 \le \lambda < 1$.

▶ **Theorem 21.** *Let $\lambda \in (0,1)$. Algorithm 1 is terminating and computes $\mathbf{d}_\lambda$.*

Undiscounted case. For $\lambda = 1$, the termination condition of the simple policy-iteration algorithm of Section 6.1 is not sufficient to guarantee correctness, since Algorithm 1 may terminate prematurely by returning a fixed point of Δ_1 that is not the minimal one.

Towards a way to obtain a stronger termination condition, we introduce the notion of self-closed relations w.r.t. a fixed point for Δ_1, originally due to [16].

▶ **Definition 22.** *A relation $M \subseteq S \times S$ is* self-closed w.r.t. $d = \Delta_1(d)$ *if, whenever $s\ M\ t$,*

- *$\ell(s) = \ell(t)$ and $d(s,t) > 0$,*
- *if $s \to \mu$ and $d(s,t) = \min_{\nu' \in \delta(t)} \mathcal{K}(d)(\mu,\nu')$ then there exists $t \to \nu$ and $\omega \in \Omega(\mu,\nu)$ such that $d(s,t) = \sum_{u,v \in S} d(u,v) \cdot \omega(u,v)$ and $supp(\omega) \subseteq M$,*
- *if $t \to \nu$ and $d(s,t) = \min_{\mu' \in \delta(s)} \mathcal{K}(d)(\mu',\nu)$ then there exists $s \to \mu$ and $\omega \in \Omega(\mu,\nu)$ such that $d(s,t) = \sum_{u,v \in S} d(u,v) \cdot \omega(u,v)$ and $supp(\omega) \subseteq M$.*

Two states are self-closed w.r.t d, written $s \approx_d t$, if they are related by some self-closed relation w.r.t. d.

It can be easily shown that $\approx_d$ is the largest self-closed relation w.r.t. d. Note that the concept of self-closeness above is defined only for fixed points of Δ_1. As remarked in [16], the largest self-closed relation $\approx_d$ can be computed in polynomial time by using partition refinement techniques similar to those employed to compute the largest bisimilarity relation.

The next lemma states that if for a fixed point $d = \Delta_1(d)$ the relation $\approx_d$ is nonempty, then d is not the least fixed point of Δ_1.

▶ **Lemma 23.** *Let $d = \Delta_1(d)$. If there exists a nonempty self-closed relation M w.r.t. d, then there exists $d_M \sqsubset d$ such that $\Delta_1(d_M) \sqsubseteq d_M$. Moreover, d_M can be computed in polynomial time in the size of the probabilistic automaton $\mathcal{A}$.*

The proof of Lemma 23 is essentially that of [16, Theorem 3]. Given a nonempty self-closed relation M w.r.t. d, the above result can be used to obtain a prefix point of Δ_1, namely d_M, that improves d towards the search of the least fixed point. The prefix point d_M of Lemma 23 is obtained from d by subtracting a constant $\theta > 0$ from all the distances computed at pairs of states in M:

$$d_M(s,t) = \begin{cases} d(s,t) - \theta & \text{if } (s,t) \in M\,, \\ d(s,t) & \text{if } (s,t) \notin M\,, \end{cases}$$

where θ is the maximal value satisfying the following set of inequalities

$$\begin{aligned} &\theta \leq d(s,t) - \min_{\nu' \in \delta(t)} \mathcal{K}(d)(\mu,\nu') && \text{for all } (s,t) \in M \text{ and } \mu \in \delta(s), \\ &\theta \leq d(s,t) - \min_{\mu' \in \delta(s)} \mathcal{K}(d)(\mu',\nu) && \text{for all } (s,t) \in M \text{ and } \nu \in \delta(t), \\ &\theta \leq d(s,t) && \text{for all } (s,t) \in M\,. \end{aligned}$$

The fact that d_M is a prefix point follows since M is a self-closed relation.

The following lemma provides us with a termination condition for the simple policy iteration algorithm to compute $\mathbf{d}_1$. Indeed, according to it, if d is a fixed point of Δ_1, we can assert that d is equal to bisimilarity distance $\mathbf{d}_1$ by simply checking that the maximal self-closed relation w.r.t. d is empty.

▶ **Lemma 24.** *If $d = \Delta_1(d)$ and $\approx_d = \emptyset$, then $d = \mathbf{d}_1$.*

Algorithm 2 extends the procedure described in Section 6.1 by encapsulating the policy iteration update (lines 4–7) into an outer-loop (lines 3–15) that is responsible to check whether the fixed point $\gamma_1^{\mathcal{C}_i}$ returned is the minimal one. According to Lemma 12, $\Delta_1(\gamma_1^{\mathcal{C}_i}) \sqsubseteq \gamma_1^{\mathcal{C}_i}$. Hence, when we reach line 8, we have that $\Delta_1(\gamma_1^{\mathcal{C}_i}) = \gamma_1^{\mathcal{C}_i}$. Therefore, by Lemmas 23 and 24, $\gamma_1^{\mathcal{C}_i} = \mathbf{d}_1$ if and only if $M = \approx_{\gamma_1^{\mathcal{C}_i}}$ is empty. If M is empty, we set the variable isMin to *true* (line 10) causing the outer-loop to terminate. Otherwise, we construct $d = (\gamma_1^{\mathcal{C}_i})_M$ as in

Algorithm 2: Simple policy iteration algorithm computing $\mathbf{d}_1$.

```
1  Initialise $\mathcal{C} = (f, \rho)$ as an arbitrary vertex coupling structure for $\mathcal{A}$
2  isMin ← false
3  while $\neg$isMin do
4      while $\exists (s,t).\, \Delta_1(\gamma_1^{\mathcal{C}})(s,t) < \gamma_1^{\mathcal{C}}(s,t)$ do
5          $R \leftarrow h(\gamma_1^{\mathcal{C}})(s,t)$
6          $\mathcal{C} \leftarrow \big(k(\gamma_1^{\mathcal{C}}),\, \rho[(s,t)/R]\big)$        /* update coupling structure */
7      end
8      Let $M \leftarrow \approx_{\gamma_1^{\mathcal{C}}}$        /* note that $\gamma_1^{\mathcal{C}} = \Delta_1(\gamma_1^{\mathcal{C}})$ */
9      if $M = \emptyset$ then
10         isMin ← true        /* $\gamma_1^{\mathcal{C}} = \mathbf{d}_1$ */
11     else
12         Compute $d = (\gamma_1^{\mathcal{C}})_M$ as in Lemma 23
13         Re-initialise $\mathcal{C}$ as a vertex coupling structure s.t. $\Gamma_1^{\mathcal{C}}(d) = \Delta_1(d)$
14     end
15 end
16 return $\gamma_1^{\mathcal{C}}$
```

Lemma 23 (line 12) and re-start the inner-loop from a vertex coupling structure $\mathcal{C}_{i+1}$ such that $\Gamma_1^{\mathcal{C}_{i+1}}(d) = \Delta_1(d)$ (line 13) (e.g., by using $\mathcal{C}_{i+1} = (k(d), \rho)$ where $\rho(s,t) = h(d)(s,t)$ for all $s, t \in S$). As proven in Theorem 25, $\gamma_1^{\mathcal{C}_i} \sqsupset \gamma_1^{\mathcal{C}_{i+1}}$. This guarantees a strict improvement of the discrepancy towards the minimal one. Termination of Algorithm 2 is justified by similar arguments as for the discounted case.

▶ **Theorem 25.** *Algorithm 2 is terminating and computes* $\mathbf{d}_1$.

6.2 Experimental Results

In this section, we evaluate the performance of the simple policy iteration algorithms on a collection of randomly generated probabilistic automata. All the algorithms have been implemented in Java and the source code is publicly available[3].

The performance of Algorithm 1 has been compared with an implementation of the *value iteration algorithm* proposed by Fu [16, Section 4]. This algorithm works as follows. Starting from the bottom element, it iteratively applies Δ_λ to the current distance function generating the increasing chain $\mathbf{0} \sqsubseteq \Delta_\lambda(\mathbf{0}) \sqsubseteq \Delta_\lambda^2(\mathbf{0}) \sqsubseteq \cdots \sqsubseteq \Delta_\lambda^{k-1}(\mathbf{0}) \sqsubseteq \Delta_\lambda^k(\mathbf{0})$.

For each input instance, the comparison involves the following steps:

1. We run Algorithm 1, storing execution time, the number of solved transportation problems, and the number of coupling structures generated during the execution (i.e., the number of times a λ-discrepancy has been computed);
2. Then, on the same instance, we execute the value iteration algorithm until the running time exceeds that of step 1. We report the execution time, the number of solved transportation problems, and the number of iterations.
3. Finally, we report the error $\max_{s,t \in S} |\mathbf{d}_\lambda(s,t) - d(s,t)|$ between the distance $\mathbf{d}_\lambda$ computed in step 1 and the approximate result d obtained in step 2.

[3] https://bitbucket.org/discoveri/probabilistic-bisimilarity-distances-probabilistic-automata

Table 1 Comparison between Simple Policy and Value Iteration Algorithm. Average performance conducted on 100 randomly generated automata with number of states $n = 10..50$, nondeterministic out-degree $k = 1..3$, and probabilistic out-degree $p = 2..3$. Discount $\lambda = 0.8$; accuracy 0.000001.

$n = \|S\|$	Simple Policy Iteration			Value Iteration			Error
	time (sec)	# TP	# $\mathcal{C}$	time (sec)	# TP	# Iter	
10	0.254	356.3	23.2	0.291	733.3	4.3	0.06005
11	0.383	454	29	0.426	987	4.8	0.04971
12	0.549	564.5	35.3	0.613	1226.5	5	0.05156
13	0.742	678.5	42.6	0.820	1480	5.5	0.05252
14	1.099	809.6	50	1.212	1877.5	5.8	0.04841
15	1.668	957.8	58.6	1.879	2160.8	6	0.05431
20	4.405	1883.3	111	6.892	4661	7.8	0.03198
30	42.392	4570.2	251.2	44.263	14625.4	11	0.02026
40	160.725	8569.5	474.5	169.457	30713.6	14.4	0.00459
50	473.598	13877	748.6	490.575	56155.2	16,8	0.01224

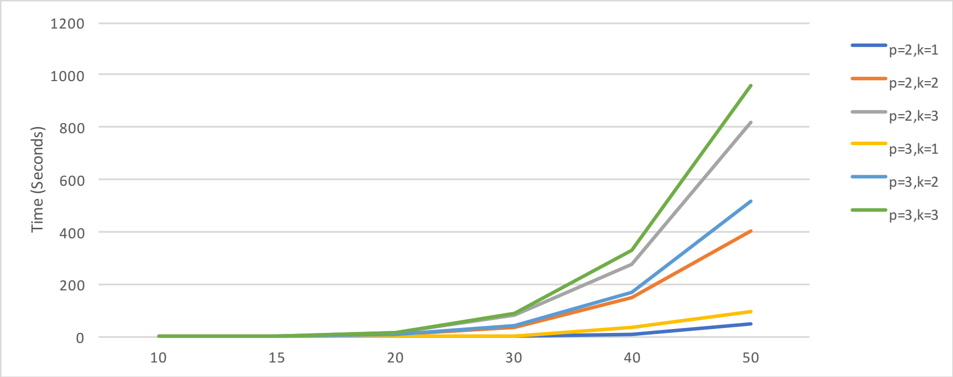


Figure 2 Average performance for the Simple Policy Iteration Algorithm conducted on 100 randomly generated automata varying number of states $n = 10..50$, nondeterministic out-degree $k = 1..3$, and probabilistic out-degree $p = 2..3$. Discount factor $\lambda = 0.8$; accuracy 0.000001.

This has been done for a collection of automata varying from 10 to 50 states. For each $n = 10, \dots, 50$, we considered 100 randomly generated probabilistic automata, varying probabilistic out-degree and nondeterministic out-degree. Table 1 reports the average results of the comparison. Our algorithm is able to compute the solution before value iteration can under-approximate it with an error ranging from 0.004 to 0.06 which is a non negligible error considering that we fixed $\lambda = 0.8$ and the distance has values in $[0, 1]$.

Furthermore in Figure 2 we observe that the execution time of the simple policy iteration algorithm is particularly influenced by the degree of nondeterminism of the automaton. This may be explained by the fact that the current implementation uses a linear program for computing the λ-discrepancy (cf. Remark 11) which has $O(n^2k^2)$ variables and $O(n^2k^2)$ constraints where n and k are the number of states and the nondeterministic out-degree of the automaton, respectively.

Algorithm 2 extends the simple policy iteration algorithms proposed in [1, 37] for Markov chains. As pointed out in [36], implementations based on the decision procedure for the existential fragment of the first-order theory of the reals fail to handle Markov chains with a handful of states. For probabilistic automata, the algorithms in [6, 7] suffer from the same

Table 2 Average performance of Algorithm 2 conducted on 100 randomly generated automata with number of states $n = 10..50$, nondeterministic out-degree $k = 1..3$, and probabilistic out-degree $p = 2..3$. Discount $\lambda = 1$; accuracy 0.000001.

$n = \lvert S \rvert$	time (sec)	# TP	# $\mathcal{C}$	# outer-loops
10	0.310	392.6	23.8	1
11	0.460	509	30.3	1
12	0.665	651.5	37.2	1
13	0.921	807	44.5	1
14	1.205	952.6	51.5	1
15	1.657	1158.2	59.8	1
20	7.633	2278.5	108.8	1
30	29.430	4880.2	207	1
40	121.855	8245.3	340.3	1
50	381.563	12938	532.5	1

problem. The performance of Algorithm 2 is comparable to that of Algorithm 1 (cf. Table 2). Despite the fact that the simple policy algorithm is not guaranteed to be sound when the discount factor equals one, our experiments show that in practice a single iteration of the outer-loop of Algorithm 2 is often sufficient to yield the correct solution.

7 Conclusion and Future Work

We presented a novel characterization of the probabilistic bisimilarity distance of Deng et al. [12] as the solution of a simple stochastic game. Starting from it, we designed algorithms for computing the distances based on Condon's simple policy iteration algorithm. The correctness of Condon's approach relies on the assumption that the input game is stopping. This may not be the case for our probabilistic bisimilarity games when the discount factor is one. We overcame this problem by means of an improved termination condition based on the notion of self-closed relation due to Fu [17].

As in [37], our simple policy iteration algorithm has exponential worst-case time complexity. Nevertheless, experiments show that our method can compete in practice with the value iteration algorithm by Fu [17] which has theoretical polynomial-time complexity for $\lambda < 1$. To the best of our knowledge, our algorithm is the first practical solution for computing the bisimilarity distance when $\lambda = 1$, performing orders of magnitude faster than the existing solutions based on the existential fragment of the first-order theory of the reals [6, 7, 9].

As future work, we plan to improve upon the current implementation in the line of [38], by exploiting the fact that bisimilar states and probabilistic distance one [39] can be efficiently pre-computed before to start the policy iteration. We believe that this would yield a significant cut down in the time required to compute the discrepancy at each iteration which turned out to be the bottleneck of our algorithms.

More efficient algorithms might lead to the speedup of verification tools for concurrent probabilistic systems, as behavioral distances relate to the satisfiability of logical properties. For the case of Markov chains, in [8, 2] the variational difference between two states with respect to their probability of satisfying linear-time properties (eg., LTL formulas) is shown to be bound by the (undiscounted) probabilistic bisimilarity distance. Some preliminary results show that a similar bound holds also for probabilistic automata, though with additional subtleties that arise by the need of resolving the non-determinism.

We also plan to extend the work on approximated minimization [3, 4] to the case of probabilistic automata and explore the possible relation between the probabilistic bisimilarity distance with more expressive logics for concurrent probabilistic systems [6, 7, 27].

References

1 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. On-the-Fly Exact Computation of Bisimilarity Distances. In *19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2013*, volume 7795 of *LNCS*, pages 1–15, 2013. doi:10.1007/978-3-642-36742-7_1.

2 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. Converging from Branching to Linear Metrics on Markov Chains. In *12th International Colloquium Theoretical Aspects of Computing, ICTAC 2015*, volume 9399 of *Lecture Notes in Computer Science*, pages 349–367. Springer, 2015. doi:10.1007/978-3-319-25150-9_21.

3 Giovanni Bacci, Giorgio Bacci, Kim G. Larsen, and Radu Mardare. On the Metric-Based Approximate Minimization of Markov Chains. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPIcs*, pages 104:1–104:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.104.

4 Giovanni Bacci, Giorgio Bacci, Kim G. Larsen, and Radu Mardare. On the metric-based approximate minimization of Markov Chains. *J. Log. Algebr. Meth. Program.*, 100:36–56, 2018.

5 Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

6 Krishnendu Chatterjee, Luca de Alfaro, Rupak Majumdar, and Vishwanath Raman. Algorithms for Game Metrics. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2008*, volume 2 of *LIPIcs*, pages 107–118. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2008. doi:10.4230/LIPIcs.FSTTCS.2008.1745.

7 Krishnendu Chatterjee, Luca de Alfaro, Rupak Majumdar, and Vishwanath Raman. Algorithms for Game Metrics (Full Version). *Logical Methods in Computer Science*, 6(3), 2010. doi:10.2168/LMCS-6(3:13)2010.

8 Di Chen, Franck van Breugel, and James Worrell. On the Complexity of Computing Probabilistic Bisimilarity. In *15th International Conference on Foundations of Software Science and Computational Structures, FOSSACS 2012*, volume 7213 of *Lecture Notes in Computer Science*, pages 437–451. Springer, 2012. doi:10.1007/978-3-642-28729-9_29.

9 Taolue Chen, Tingting Han, and Jian Lu. On Behavioral Metric for Probabilistic Systems: Definition and Approximation Algorithm. In *4th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2007*, pages 21–25. IEEE Computer Society, 2007. doi:10.1109/FSKD.2007.426.

10 Anne Condon. On Algorithms for Simple Stochastic Games. In *Advances In Computational Complexity Theory*, volume 13 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 51–72. DIMACS/AMS, 1990.

11 Anne Condon. The Complexity of Stochastic Games. *Inf. Comput.*, 96(2):203–224, 1992. doi:10.1016/0890-5401(92)90048-K.

12 Yuxin Deng, Tom Chothia, Catuscia Palamidessi, and Jun Pang. Metrics for Action-labelled Quantitative Transition Systems. *Electr. Notes Theor. Comput. Sci.*, 153(2):79–96, 2006. doi:10.1016/j.entcs.2005.10.033.

13 Josee Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for Labelled Markov Processes. *Theor. Comput. Sci.*, 318(3):323–354, 2004. doi:10.1016/j.tcs.2003.09.013.

14 Josée Desharnais, François Laviolette, and Mathieu Tracol. Approximate Analysis of Probabilistic Processes: Logic, Simulation and Games. In *5th International Conference on the Quantitative Evaluaiton of Systems, QEST 2008*, pages 264–273. IEEE Computer Society, 2008. doi:10.1109/QEST.2008.42.

15 Nathanaël Fijalkow, Bartek Klin, and Prakash Panangaden. Expressiveness of Probabilistic Modal Logics, Revisited. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPIcs*, pages 105:1–105:12. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.105.

16 Hongfei Fu. Computing Game Metrics on Markov Decision Processes. In *39th International Colloquium on Automata, Languages, and Programming, ICALP 2012*, volume 7392 of *Lecture Notes in Computer Science*, pages 227–238. Springer, 2012. doi:10.1007/978-3-642-31585-5_23.

17 Hongfei Fu. personal communication, 2013.

18 Alessandro Giacalone, Chi-Chang Jou, and Scott A. Smolka. Algebraic Reasoning for Probabilistic Concurrent Systems. In *Proceedings of the IFIP WG 2.2/2.3 Working Conference on Programming Concepts and Methods*, pages 443–458. North-Holland, 1990.

19 Felix Hausdorff. *Grundzüge der Mengenlehre.* Verlag Von Veit & Comp, Leipzig, 1914.

20 Bengt Jonsson and Kim G. Larsen. Specification and Refinement of Probabilistic Processes. In *6th Annual Symposium on Logic in Computer Science, LICS 1991*, pages 266–277. IEEE Computer Society, 1991. doi:10.1109/LICS.1991.151651.

21 Brendan Juba. On the Hardness of Simple Stochastic Games. Master's thesis, Carnegie Mellon University, Pittsburgh, PA, USA, May 2005.

22 Leonid Vitalevich Kantorovich. On the transfer of masses (in Russian). *Doklady Akademii Nauk*, 5(5-6):1–4, 1942. Translated in Management Science, 1958.

23 Peter Kleinschmidt and Heinz Schannath. A Strongly Polynomial Algorithm for the Transportation Problem. *Math. Program.*, 68:1–13, 1995. doi:10.1007/BF01585755.

24 Barbara König and Christina Mika-Michalski. (Metric) Bisimulation Games and Real-Valued Modal Logics for Coalgebras. In *29th International Conference on Concurrency Theory, CONCUR 2018*, volume 118 of *LIPIcs*, pages 37:1–37:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.37.

25 Thomas Liggett and Steven A. Lippman. Stochastic Games with Perfect Information and Time Average Payoff. *SIAM Review*, 11(4):604–607, 1969. doi:10.1137/1011093.

26 Facundo Mémoli. Gromov-Wasserstein Distances and the Metric Approach to Object Matching. *Foundations of Computational Mathematics*, 11(4):417–487, 2011. doi:10.1007/s10208-011-9093-5.

27 Matteo Mio. On the Equivalence of Game and Denotational Semantics for the Probabilistic μ-calculus. *Logical Methods in Computer Science*, 8(2), 2012. doi:10.2168/LMCS-8(2:7)2012.

28 James B. Orlin. *On the Simplex Algorithm for Networks and Generalized Networks*, pages 166–178. Springer Berlin Heidelberg, 1985. doi:10.1007/BFb0121050.

29 Gabriel Peyré and Marco Cuturi. Computational Optimal Transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019. doi:10.1561/2200000073.

30 Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1994.

31 Alexander Schrijver. *Theory of Linear and Integer Programming.* Wiley-Interscience series in Discrete Mathematics and Optimization. Wiley, 1999.

32 Roberto Segala. *Modeling and Verification of Randomized Distributed Real-time Systems.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995.

33 Roberto Segala and Nancy A. Lynch. Probabilistic Simulations for Probabilistic Processes. In *5th International Conference on Concurrency Theory, CONCUR 1994*, volume 836 of *Lecture Notes in Computer Science*, pages 481–496. Springer, 1994. doi:10.1007/978-3-540-48654-1_35.

34 Lloyd S. Shapley. Stochastic Games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100, 1953. doi:10.1073/pnas.39.10.1095.

35 James K. Strayer. *Linear Programming and its Applications.* Undergraduate Texts in Mathematics. Springer-Verlag, New York, NY, USA, 1989. doi:10.1007/978-1-4612-1009-2.

36 Qiyi Tang. *Computing Probabilistic Bisimilarity Distances.* PhD thesis, York University, Toronto, Canada, August 2018.

37 Qiyi Tang and Franck van Breugel. Computing Probabilistic Bisimilarity Distances via Policy Iteration. In *27th International Conference on Concurrency Theory, CONCUR 2016*, volume 59 of *LIPIcs*, pages 22:1–22:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.22.

38 Qiyi Tang and Franck van Breugel. Deciding Probabilistic Bisimilarity Distance One for Labelled Markov Chains. In *30th International Conference on Computer Aided Verification, CAV 2018*, volume 10981 of *Lecture Notes in Computer Science*, pages 681–699. Springer, 2018. doi:10.1007/978-3-319-96145-3_39.

39 Qiyi Tang and Franck van Breugel. Deciding Probabilistic Bisimilarity Distance One for Probabilistic Automata. In Sven Schewe and Lijun Zhang, editors, *29th International Conference on Concurrency Theory, CONCUR 2018*, volume 118 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 9:1–9:17. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. doi:10.4230/LIPIcs.CONCUR.2018.9.

40 Franck van Breugel and James Worrell. The Complexity of Computing a Bisimilarity Pseudometric on Probabilistic Automata. 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 *Lecture Notes in Computer Science*, pages 191–213. Springer, 2014. doi:10.1007/978-3-319-06880-0_10.

41 Cédric Villani. *Optimal Transport: Old and New.* Grundlehren der mathematischen Wissenschaften. Springer, 2008.

Asymmetric Distances for Approximate Differential Privacy

Dmitry Chistikov
Centre for Discrete Mathematics and its Applications (DIMAP) & Department of Computer Science, University of Warwick, UK

Andrzej S. Murawski
Department of Computer Science, University of Oxford, UK

David Purser
Centre for Discrete Mathematics and its Applications (DIMAP) & Department of Computer Science, University of Warwick, UK

Abstract

Differential privacy is a widely studied notion of privacy for various models of computation, based on measuring differences between probability distributions. We consider (ϵ, δ)-differential privacy in the setting of labelled Markov chains. For a given ϵ, the parameter δ can be captured by a variant of the total variation distance, which we call lv_α (where $\alpha = e^\epsilon$).

First we study lv_α directly, showing that it cannot be computed exactly. However, the associated approximation problem turns out to be in **PSPACE** and #**P**-hard. Next we introduce a new bisimilarity distance for bounding lv_α from above, which provides a tighter bound than previously known distances while remaining computable with the same complexity (polynomial time with an **NP** oracle). We also propose an alternative bound that can be computed in polynomial time. Finally, we illustrate the distances on case studies.

2012 ACM Subject Classification Theory of computation → Probabilistic computation

Keywords and phrases Bisimilarity distances, Differential privacy, Labelled Markov chains

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.10

Funding *Andrzej S. Murawski*: Royal Society Leverhulme Trust Senior Research Fellowship and the International Exchanges Scheme (IE161701)
David Purser: UK EPSRC Centre for Doctoral Training in Urban Science (EP/L016400/1)

Acknowledgements The authors would like to thank the reviewers for their helpful comments.

1 Introduction

Differential privacy [14] is a security property that ensures that a small perturbation of the input leads to only a small perturbation in the output, so that observing the output makes it difficult to discern whether a particular piece of information was present in the input. It has been shown that various bisimilarity distances can bound the differential privacy of a labelled Markov chain, by bounding for example the ϵ [6, 31] and δ [9] privacy parameters. Bisimilarity distances [17, 11] were introduced as a metric analogue of probabilistic bisimulation [23], to overcome the problem that bisimilarity is too sensitive to minor changes in probabilities.

We further the study of bounds to δ by defining new bisimilarity distances. The bisimilarity distance of [9], inspired by the work of [31], transpired to be computable in polynomial time with an **NP** oracle. The work of [31] defined distances using the Kantorovich metric and the associated bisimilarity distance based on a fixed point; and considered the effect of replacing the absolute value function with another metric. For the purposes of (ϵ, δ)-differential privacy the distance required is not a metric, nor even a pseudometric, so their methods are adapted in [9] to account for this; resulting in a distance function bd_α which can be used to bound

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 10; pp. 10:1–10:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

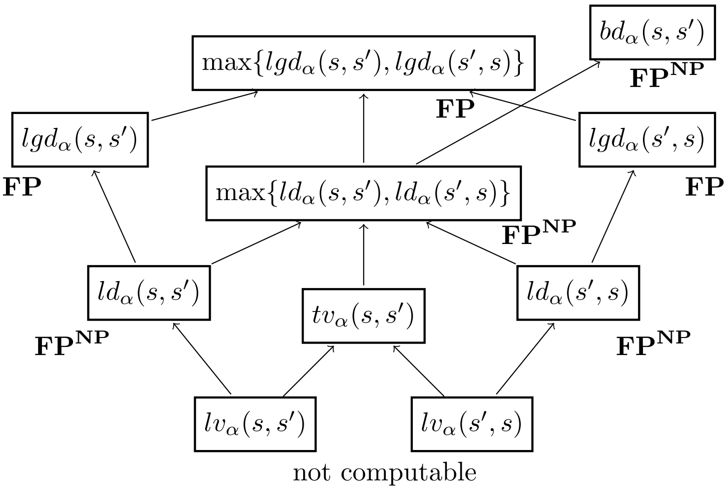


Figure 1 Partial order of distances, such that $a \to b \iff a \leq b$. **FP** is the functional counterpart of **P**, where the value of the function can be computed in polynomial time. $\mathbf{FP^{NP}}$ indicates polynomial time with **NP** oracle. tv_α and bd_α are introduced in [9] and recalled in Sections 3 and 6, respectively. The remaining distances are the contribution of this paper.

the δ parameter in differential privacy from above. The function, however, retained the symmetry property that $bd_\alpha(s,s') = bd_\alpha(s',s)$. In this paper we further study distances to bound differential privacy in labelled Markov chains, but drop this symmetry property and discover a tighter bound, which can be computed with the same cost. We also define a weaker bisimilarity distance for bounding δ that can be computed in polynomial time.

The privacy parameter in question, δ, can be expressed as a variant of the total variation distance tv_α. In particular we define lv_α as a single component of tv_α (which is a maximum over two functions). This distance is a way of measuring the maximum difference of probabilities between any two states. Total variation distance is usually expressed using absolute difference, but for differential privacy a skew is introduced into this distance. These exact distances transpire to be very difficult to compute: we confirm that the threshold distance problem, which asks whether the distance is below a given threshold, is undecidable and approximating it is #**P**-hard. We also show that for finite words it can be approximated in **PSPACE**. These results match the results of [22] for standard total variation distances.

We then bound the distance lv_α from above by a distance ld_α which will turn out to be computable, in a similar manner to how bd_α bounds tv_α in [9]. We show that ld_α can be computed in polynomial time with an **NP** oracle (that is, with the same complexity as bd_α). We further generalise ld_α to a new distance lgd_α, computable in polynomial time. This new distance, is no smaller than ld_α, and we conjecture it might be equal. We can then take $\max\{ld_\alpha(s,s'), ld_\alpha(s',s)\}$ and $\max\{lgd_\alpha(s,s'), lgd_\alpha(s',s)\}$ as sound upper bounds on δ. Thus we have defined the first non-trivial estimate of the δ parameter that can be computed in polynomial time (trivially, always returning 1 is technically correct). Our results show that taking the maximum over two ld_α is a better approximation than bd_α from [9]. We confirm this using several case studies, where we also demonstrate, on a randomised response mechanism, that the estimates based on ld_α can beat standard differential privacy composition theorems. The relationships between distances are summarised in Figure 1.

Research into behavioural pseudometrics has a long history going back to Giacalone et al. [17]. Our work lies in the tradition of bisimulation pseudometrics based on the Kantorovich distance started by Desharnais et al. [11, 12], and builds upon subsequent work on computing them [29]. Chatzikokolakis et al. [6] generalised the pseudometric framework to handle ϵ-differential privacy, and indeed arbitrary metrics, but did not consider the complexity of calculating the distances. We introduced a distance in [9] for (ϵ, δ)-differential privacy, which is improved upon in this paper. As concerns approximation, we are not aware of any related work on distances other than the total variation distance [8, 22].

2 Preliminaries

Given a finite set X, let $\mathit{Dist}(X)$ be the set of all stochastic vectors in $\mathbb{R}^X$. If X is a set of symbols then X^* is the set of all sequences of symbols in X, X^+ all sequences of length at least one, and X^ω all infinite sequences.

▶ **Definition 1** (labelled Markov chains (LMC's)). *A* labelled Markov chain $\mathcal{M}$ *is a tuple* $\langle S, \Sigma, \mu, \ell \rangle$*, where* S *is a finite set of states,* Σ *is a finite alphabet,* $\mu : S \to \mathit{Dist}(S)$ *is the transition function and* $\ell : S \to \Sigma$ *is the labelling function.*

We assume that all transition probabilities are rational, represented as a pair of binary integers. *size*$(\mathcal{M})$ is the number of bits required to represent $\langle S, \Sigma, \mu, \ell \rangle$, including the bit size of the probabilities. We will write μ_s for $\mu(s)$.

In what follows, we study probabilities associated with infinite sequences of labels generated by LMC's. We specify the relevant probability spaces next using standard measure theory [5, 2]. Let us start with the definition of cylinder sets.

▶ **Definition 2.** *A subset* $C \subseteq \Sigma^\omega$ *is a* cylinder set *if there exists* $u \in \Sigma^*$ *such that* C *consists of all infinite sequences from* Σ^ω *whose prefix is* u*. We then write* C_u *to refer to* C*.*

Cylinder sets play a prominent role in measure theory in that their finite unions can be used as a generating family (an algebra) for the set $\mathcal{F}_\Sigma$ of measurable subsets of Σ^ω (the cylindrical σ-algebra). Where clear from context we will omit Σ in the subscript of $\mathcal{F}$. What will be important for us is that any measure ν on $(\Sigma^\omega, \mathcal{F}_\Sigma)$ is uniquely determined by its values on cylinder sets [5, Chapter 1, Section 2][2, Section 10.1]. Next we show how to assign a measure ν_s on $(\Sigma^\omega, \mathcal{F}_\Sigma)$ to an arbitrary state of an LMC $\mathcal{M}$.

▶ **Definition 3.** *Given* $\mathcal{M} = \langle S, \Sigma, \mu, \ell \rangle$*, let* $\mu^+ : S^+ \to [0, 1]$ *and* $\ell^+ : S^+ \to \Sigma^+$ *be the natural extensions of the functions* μ *and* ℓ *to* S^+*, i.e.* $\mu^+(s_0 \cdots s_k) = \prod_{i=0}^{k-1} \mu_{s_i}(s_{i+1})$ *and* $\ell^+(s_0 \cdots s_k) = \ell(s_0) \cdots \ell(s_k)$*, where* $k \geq 0$ *and* $s_i \in S$ $(0 \leq i \leq k)$*. Note that, for any* $s \in S$*, we have* $\mu^+(s) = 1$*. Given* $s \in S$*, let* $\mathit{Paths}_s(\mathcal{M})$ *be the subset of* S^+ *consisting of all sequences that start with* s*.*

▶ **Definition 4.** *Let* $\mathcal{M} = \langle S, \Sigma, \mu, \ell \rangle$ *and* $s \in S$*. We define* $\nu_s : \mathcal{F}_\Sigma \to [0, 1]$ *to be the unique measure on* $(\Sigma^\omega, \mathcal{F}_\Sigma)$ *such that for any cylinder* C_u *we have* $\nu_s(C_u) = \sum \mu^+(p)$ *where the summation is over* $p \in \mathit{Paths}_s(\mathcal{M})$ *such that* $\ell^+(p) = u$*.*

▶ **Example 5** (transition-labelled LMC's). Like in [29, 7, 1, 27, 9], Definition 1 features labelled states. However, Markov chains with labelled transitions can also be described in the framework of that definition.

In particular, suppose we are given a chain $\mathcal{M}$ of the form $\langle S, \Sigma, T \rangle$, where S is a finite set of states, Σ is a finite alphabet and $T : S \to \mathit{Dist}(S \times \Sigma)$ is the transition function. We write each transition as $q \xrightarrow[a]{p} q'$, meaning that $T(q)(q', a) = p$. From this transition-labelled LMC, we create an equivalent state-labelled Markov chain $\mathcal{M}'$: for each state and each label, add new state (q, a) labelled with a, such that, when $q \xrightarrow[b]{p} q'$, we have $\mu_{(q,a)}((q', b)) = p$ for every

$a \in \Sigma$. Technically, this delays reading of the first character until the second state visited. To account for this, introduce an additional character, say $\vdash$, so that $\nu_s(C_w) = \nu'_{(s,\vdash)}(C_{\vdash w})$, where ν and ν' refer to the measures associated with $\mathcal{M}$ and $\mathcal{M}'$ respectively (Definition 4).

▶ **Example 6** (finite-word LMC's)**.** We can also describe labelled Markov chains over finite words. These chains have a set of final states F, which have no outgoing transitions. We require positive probability of reaching a final state from every reachable state. We define the function $\nu_s(w) = \sum \mu^+(p)$, where the summation is over $p \in \mathit{Paths}_s(\mathcal{M})$ such that $\ell^+(p) = w$ and $p_{|w|} \in F$, so that we only consider paths which end in a final state. The function can be extended to sets of words $E \subseteq \Sigma^*$ (which are countable) by $\nu_s(E) = \sum_{w \in E} \nu_s(w)$.

Such machines can also be represented by infinite-word Markov chains. One can simulate the end of the word by an additional character, say \$ such that, for $q \in F$, $\mu_q(q) = 1$ and $\ell(q) = \$$, so that the only trace that can be observed from q is $\$^\omega$. Then, for a word $w \in \Sigma^*$, we rather study $w\$\$\$\dots$, corresponding to the cylinder $C_{w\$}$. In the translated infinite-word model, the event C_u corresponds to the event $\{w \in \Sigma^* \mid \mathit{prefix}(w) = u\}$ in the original finite-word model. Some of our arguments will be carried out in the finite-word setting, as hardness results that apply to these chains also apply to infinite-word Markov chains. Other arguments will only be possible in the finite-word setting.

Let us return to the general definition of Markov chains (Definition 1). Our aim will be to compare states from the point of view of differential privacy. Any two states s, s' can be viewed as indistinguishable if $\nu_s(E) = \nu_{s'}(E)$ for every $E \in \mathcal{F}$. More generally, the difference between them can be quantified using the *total variation distance*, defined by $\mathit{tv}(\nu, \nu') = \sup_{E \in \mathcal{F}} |\nu(E) - \nu'(E)|$. Given $\mathcal{M} = \langle S, \Sigma, \mu, \ell\rangle$ and $s, s' \in S$, we shall write $\mathit{tv}(s, s')$ to refer to $\mathit{tv}(\nu_s, \nu_{s'})$. Ensuring such pairs of measures $(\nu_s, \nu_{s'})$ are “similar” is essential for privacy, so that it is difficult to observe which of the states was the originating position. To measure probabilities relevant to differential privacy, we will need to study a more general variant lv_α of the above distance, which we introduce shortly.

3 (ϵ, δ)-Differential Privacy

Differential privacy is a mathematically rigorous definition of privacy due to Dwork *et al* [14]; the aim is to ensure that inputs which are related in some sense lead to very similar outputs. Formally it requires that for two related states there only ever be a small change in output probabilities, and therefore discerning which of the two states was actually used is difficult, maintaining their privacy. We rely on the definition of *approximate differential privacy* in the context of labelled Markov chains, as per [9].

▶ **Definition 7.** *Let $\mathcal{M} = \langle S, \Sigma, \mu, \ell\rangle$ be a labelled Markov chain and let $R \subseteq S \times S$ be a symmetric relation. Given $\epsilon \geq 0$ and $\delta \in [0, 1]$, we say that $\mathcal{M}$ is (ϵ, δ)-differentially private w.r.t. R if, for every $s, s' \in S$ such that $(s, s') \in R$, we have $\nu_s(E) \leq e^\epsilon \cdot \nu_{s'}(E) + \delta$ for every measurable set $E \in \mathcal{F}$.*

What it means for two states to be related, as specified by R, is to a large extent domain-specific. In general, R makes it possible to spell out which states should not appear too different and, consequently, should enjoy a quantitative amount of privacy.

Note that each state $s \in S$ can be viewed as defining a random variable X_s with outcomes from Σ^ω such that $\mathbb{P}[X_s \in E] = \nu_s(E)$. Then the above can be rewritten as $\mathbb{P}[X_s \in E] \leq e^\epsilon\, \mathbb{P}[X_{s'} \in E] + \delta$, which matches the definition from [14], where one would consider $X_s, X_{s'}$ neighbouring in some natural sense. In the typical database scenario, one would relate database states that differ by exactly one entry. In our setting, we refer to states of a machine, for which we would like it to be indiscernible as to which was the start state, assuming that the states are hidden and the traces are observable.

When $\delta = 0$, we use the term *ϵ-differential privacy*, which amounts to measuring the ratio between the probabilities of possible outcomes. When one cannot expect to achieve this pure *ϵ-differential privacy*, the relaxed approximate differential privacy is used [24]. When $\epsilon = 0$, δ is captured exactly by the statistical distance (total variation distance) tv.

Our aim is to capture the value of δ required to satisfy the differential privacy property for a given ϵ. That is, given a LMC $\mathcal{M}$, a symmetric relation R and $\alpha = e^{\epsilon} \geq 1$, we want to determine the smallest δ such that $\mathcal{M}$ is (ϵ, δ)-differentially private with respect to R. We can measure the difference between two measures ν, ν' on $(\Sigma^{\omega}, \mathcal{F})$ as follows: $tv_{\alpha}(\nu, \nu') = \sup_{E \in \mathcal{F}} \Delta_{\alpha}(\nu(E), \nu'(E))$ where $\Delta_{\alpha}(a, b) = \max\{a - \alpha b, b - \alpha a, 0\}$ [3]. When used on $\nu_s, \nu_{s'}$ and $\alpha = e^{\epsilon}$, $tv_{\alpha}(s, s')$ gives the required δ between states s, s' [9].

In this paper we observe that significant simplification occurs by splitting the two main parts of the maximum, taking only the "left variant". Whilst Δ_{α} is symmetric, we break this property to introduce a new distance function Λ_{α} (similarly to [4]). Then we define an analogous total variation distance lv_{α}, which will be our main object of study.

▶ **Definition 8** (Asymmetric skewed total variation distance)**.** *Let $\alpha \geq 1$. Given two measures ν, ν' on $(\Sigma^{\omega}, \mathcal{F})$, let $lv_{\alpha}(\nu, \nu') = \sup_{E \in \mathcal{F}} \Lambda_{\alpha}(\nu(E), \nu'(E))$, where $\Lambda_{\alpha}(a, b) = \max\{a - \alpha b, 0\}$.*

We will write $lv_{\alpha}(s, s')$ for $lv_{\alpha}(\nu_s, \nu_{s'})$. Note that it is not required to take the maximum with zero, that is $lv_{\alpha}(\nu, \nu') = \sup_{E \in \mathcal{F}} \nu(E) - \alpha \nu'(E)$, since there is always an event such that $\nu'(E) = 0$, in particular $\nu(\emptyset) = 0$. Observe that Δ_{α} and Λ_{α} are not metrics as $\Delta_{\alpha}(a, b) = 0 \not\Longrightarrow a = b$, and in fact not even pseudometrics as the triangle inequality does not hold. Our new distance Λ_{α} (and lv_{α}) is not symmetric, while Δ_{α} and tv_{α} are.

If $\alpha = 1$, then $lv_1 = tv_1 = tv$, since if ν, ν' are probability measures and we have $\nu(E) = 1 - \nu(\overline{E})$ then $\sup_{E \in \mathcal{F}} |\nu(E) - \nu'(E)| = \sup_{E \in \mathcal{F}} \nu(E) - \nu'(E) = \sup_{E \in \mathcal{F}} \nu'(E) - \nu(E)$, i.e., despite the use of the absolute value in the definition of tv, it is not required.

We can reformulate differential privacy in terms of tv_{α} and lv_{α}.

▶ **Proposition 9.** *Given a labelled Markov chain $\mathcal{M}$ and a symmetric relation $R \subseteq S \times S$, the following properties are equivalent for $\alpha = e^{\epsilon}$:*

- *$\mathcal{M}$ is (ϵ, δ)-differentially private w.r.t. R,*
- *$\max_{(s,s') \in R} tv_{\alpha}(s, s') \leq \delta$, and*
- *$\max_{(s,s') \in R} lv_{\alpha}(s, s') \leq \delta$.*

We now focus on computing lv_{α}, since this will allow us to determine the "level" of differential privacy for a given ϵ. Henceforth we will refer to e^{ϵ} as α. For the purposes of our complexity arguments, we will only use rational α with $O(\mathit{size}(\mathcal{M}))$-bit representation.

4 lv_{α} is not computable

$tv(s, s')$ turns out to be surprisingly difficult to compute: the threshold distance problem (whether the distance is strictly greater than a given threshold) is undecidable, and the non-strict variant of the problem ("greater or equal") is not known to be decidable [22]. The undecidability result is shown by reduction from the emptiness problem for probabilistic automata to the threshold distance problem for finite-word transition-labelled Markov chains. Recall that such chains are a special case of our more general definition of infinite-word state-labelled Markov chains. Thus, the problem is undecidable in this case also.

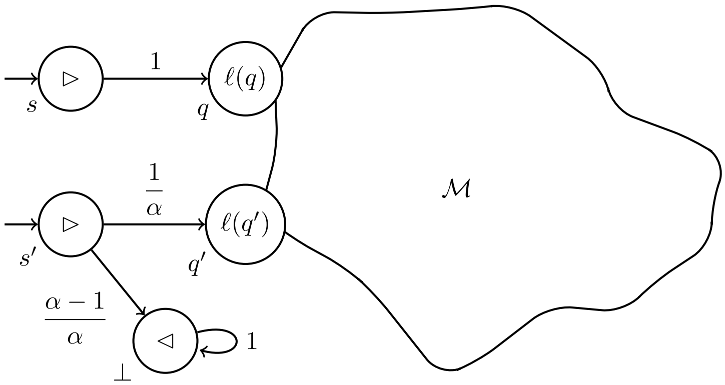


Figure 2 Markov chain $\mathcal{M}'$ in the reduction from $tv(q,q')$ to $lv_\alpha(s,s')$.

Since $tv = lv_1$, we know that $lv_1(s,s') > \theta$ is undecidable. We show that this is not special, that is, the problem remains undecidable for any fixed $\alpha > 1$. In other words, no value of the privacy parameter ϵ makes it possible to compute the optimal δ exactly.

▶ **Theorem 10.** *Finding a value of tv reduces in polynomial time to finding a value of lv_α.*

Proof. Given a labelled Markov chain $\mathcal{M} = \langle Q, \Sigma, \mu, \ell\rangle$, and states q, q' for which we require the answer $tv(q,q')$, we construct a new labelled Markov chain $\mathcal{M}'$, for which $lv_\alpha(s,s') = tv(q,q')$.

We define $\mathcal{M}' = \langle Q \cup \{s, s', \bot\}, \Sigma', \mu', \ell'\rangle$, with $\ell'(s) = \ell'(s') = \triangleright$, $\ell'(\bot) = \triangleleft$, $\ell'(x) = \ell(x)$ for all $x \in Q$, $\Sigma' = \Sigma \cup \{\triangleright, \triangleleft\}$,

$$\mu'_s(q) = 1, \qquad \mu'_{s'}(q') = \frac{1}{\alpha}, \qquad \mu'_{s'}(\bot) = \frac{\alpha-1}{\alpha}, \qquad \text{and} \qquad \mu'_x(y) = \mu_x(y) \text{ for all } x, y \in Q.$$

The reduction, sketched in Figure 2, adds three new states, so can be done in polynomial time. We claim $lv_\alpha(s,s') = tv(q,q')$.

Consider $E \in \mathcal{F}_\Sigma$, observe that $\nu_q(E) = \nu_s(E')$ and $\nu_{q'}(E) = \alpha\nu_{s'}(E')$, where $E' = \{\triangleright w \mid w \in E\} \in \mathcal{F}_{\Sigma'}$. Then $\nu_q(E) - \nu_{q'}(E) = \nu_s(E') - \alpha\nu_{s'}(E')$ and $lv_\alpha(s,s') \geq tv(q,q')$.

Conversely, consider an event $E' \in \mathcal{F}_{\Sigma'}$. Since the character $\triangleleft$ can only be reached from s', any word using it contributes negatively to the difference. Hence intersecting the event with $\triangleright\Sigma^\omega$, to remove $\triangleleft$, can only increase the difference. The character $\triangleright$ must occur (only) as the first character of every (useful) word in E'. Let $E = \{w \mid \triangleright w \in E' \cap \triangleright\Sigma^\omega\} \in \mathcal{F}_\Sigma$, then $\nu_q(E) - \nu_{q'}(E) \geq \nu_s(E') - \alpha\nu_{s'}(E')$. Thus $tv(q,q') \geq lv_\alpha(s,s')$. ◀

Since an oracle to solve decision problems for lv_α would solve problems for tv, we obtain the following result.

▶ **Corollary 11.** *$lv_\alpha(s,s') > \theta$ is undecidable for $\alpha \geq 1$.*

It is not clear that lv_α reduces easily to tv. Arguments along the lines of the proof of Theorem 10 may not result in a Markov chain due to non-stochastic transitions, or modifications to the $s \to q$ branch may result in new maximising events.

5 Approximation of lv_α

Given that lv_α cannot be computed exactly, we turn to approximation: the problem, given $\gamma > 0$, of finding some x such that $|x - lv_\alpha(s,s')| \le \gamma$. For $\alpha = 1$, it is known that approximating $tv = lv_1$ is possible in **PSPACE** but #**P**-hard [8, 22]. We show that the case $\alpha = 1$ is not special; that is, when $\alpha > 1$, lv_α can also be approximated and the same complexity bounds apply.

▶ Remark. Typically one might suggest being ϵ close ($|x - lv_\alpha(s,s')| \le \epsilon$). To avoid confusion with the differential privacy parameter, we refer to γ close.

▶ **Theorem 12.** *For finite-word Markov chains, approximation of $lv_\alpha(s,s')$ within γ can be performed in* **PSPACE** *and is* #**P**-*hard.*

Proof (sketch). For the upper bound, we show that the i^{th} bit of an x such that $|x - lv_\alpha(s,s')| \le \gamma$ can be found in **PSPACE**. The approach, inspired by [22], is to consider the maximising event of $lv_\alpha(s,s') = \sup_{E \subseteq \Sigma^*} \nu_s(E) - \alpha\nu_{s'}(E)$, which turns out to be $W = \{w \mid \nu_s(w) \ge \alpha\nu_{s'}(w)\}$, so that $lv_\alpha(s,s') = \nu_s(W) - \alpha\nu_{s'}(W)$. This choice of the maximising event only applies to finite-word Markov chains, thus the proof does not extend in full generality to infinite-word Markov chains. The shape of the event is the key difference between our proof and [22], which uses events of the form $\{w \mid \nu_s(w) \ge \nu_{s'}(w)\}$.

Let $\overline{W}$ denote the complement of W and let $\nu_s(\overline{W})$ be approximated by a number X and $\nu_{s'}(W)$ by a number Y. Normally, one would expect X to be close to $\nu_s(\overline{W})$ and Y to be close to $\nu_{s'}(W)$. Here, the trick is to require only that $\nu_s(\overline{W}) + \alpha\nu_{s'}(W)$ be close to $X + \alpha Y$. It is then argued that, for specific X, Y with this property, one can find any bit of $X + \alpha Y$.

For the lower bound, we note that approximating tv is #**P**-hard [22], by a reduction from #*NFA*, a #**P**-complete problem [20]. That is, given a non-deterministic finite automaton $\mathcal{A}$ and $n \in \mathbb{N}$ in unary, determine $|\Sigma^n \cap L(\mathcal{A})|$, the number of accepted words of $\mathcal{A}$ of length n. Since tv can be reduced to lv_α (Theorem 10), approximating lv_α is #**P**-hard as well. The hardness result applies to finite-word transition-labelled Markov chains, thus also to the more general infinite-word labelled Markov chains. ◀

6 A least fixed point bound ld_α

We seek to bound lv_α from above by a computable quantity, and will introduce a distance function ld_α for this. We first introduce a variant of the Kantorovich lifting as a technique to measure the distance between probability distributions on a set X, given a distance function between objects of X. We show that lv_α can be reformulated using such a distance over the (infinite) trace distributions $\nu_s, \nu_{s'}$. We then define an alternative distance function between states, ld_α, as the fixed point of the Kantorovich lifting of distances from individual states to (finite) state distributions. We will observe that it is possible to compute and acts as a sound bound on lv_α.

We use this distance to determine (ϵ, δ)-differential private w.r.t. relation R by bounding δ with $\max_{(s,s') \in R} ld_\alpha(s,s')$. We will show this can be achieved in polynomial time with access to an **NP** oracle, by computing $ld_\alpha(s,s')$ exactly in this time ($|R|$ is polynomial with respect to the size of $\mathcal{M}$). This suggests a complexity lower than approximation (which is #**P**-hard by Theorem 12).

▶ **Definition 13** (Asymmetric Skewed Kantorovich Lifting). *For a set X, given $d : X \times X \to [0,1]$ a distance function and measures μ, μ', we define*

$$K^{\Lambda}_{\alpha}(d)(\mu,\mu') = \sup_{\substack{f : X \to [0,1] \\ \forall x,x' \in X \;\; \Lambda_\alpha(f(x),f(x')) \leq d(x,x')}} \Lambda_\alpha\Big(\int_X f d\mu, \int_X f d\mu'\Big)$$

where f ranges over functions which are measurable w.r.t. μ and μ'.

▶ Remark. The (standard) Kantorovich distance lifts a distance function d over the ground objects X to a distance between measures μ, μ' on the set X. This is equivalent to replacing Λ_α with the absolute distance function ($abs(a,b) = |a-b|$). We note that $K^{\Lambda}_{\alpha}(d)$ is equivalent to the standard Kantorovich distance for $\alpha = 1$ and d symmetric [21, 10]. If $|X| < \infty$ (for example when X is a finite set of states, S), we have $\int_X f d\mu = \sum_{x \in X} f(x)\,\mu(x)$. Chatzikokolakis et al. [6] considered the case where the absolute value function was replaced by any metric d'. Our lifting K^{Λ}_{α} does not quite fit in this framework, since Λ_α is not metric.

The interest in K^{Λ}_{α} is that it allows us to reformulate the definition of the distance function lv_α. Our goal is to measure the difference between measures over infinite traces $\nu_s, \nu_{s'}$, and so we lift a distance function over infinite words ($d : \Sigma^\omega \times \Sigma^\omega \to [0,1]$). In particular, we lift the discrete metric $\mathbb{1}_{\neq}$ (the indicator function over inequality with $\mathbb{1}_{\neq}(w,w') = 1$ for $w \neq w'$, and 0 otherwise).

▶ **Lemma 14.** *$lv_\alpha(s,s') = K^{\Lambda}_{\alpha}(\mathbb{1}_{\neq})(\nu_s, \nu_{s'})$.*

Since computing lv_α, or now $K^{\Lambda}_{\alpha}(\mathbb{1}_{\neq})(\nu_s, \nu_{s'})$, is difficult, we introduce an upper bound on lv_α, inspired by bisimilarity distances, which we will call ld_α. This will be the least fixed point of $\Gamma^{\Lambda}_{\alpha}$, a function which measures (relative to a distance function d) the distance between the transition distributions of s, s' where s, s' share a label, or 1 when they do not.

▶ **Definition 15.** *Let $\Gamma^{\Lambda}_{\alpha} : [0,1]^{S \times S} \to [0,1]^{S \times S}$ be defined as follows.*

$$\Gamma^{\Lambda}_{\alpha}(d)(s,s') = \begin{cases} K^{\Lambda}_{\alpha}(d)(\mu_s, \mu_{s'}) & \ell(s) = \ell(s') \\ 1 & \textit{otherwise} \end{cases}$$

The utility of this function is that we are not now using the Kantorovich lifting over infinite trace distributions, but rather over finite transition distributions ($\mu_s \in Dist(S)$).

Note that $[0,1]^{S \times S}$ equipped with the pointwise order, written $\sqsubseteq$, is a complete lattice and that Γ_α is monotone with respect to that order (larger d permit more functions, thus larger supremum). Consequently, $\Gamma^{\Lambda}_{\alpha}$ has a least fixed point [28]. We take our distance to be exactly that point.

▶ **Definition 16.** *Let $ld_\alpha : S \times S \to [0,1]$ be the least fixed point of $\Gamma^{\Lambda}_{\alpha}$.*

To provide a guarantee of privacy we require a sound upper bound on lv_α.

▶ **Theorem 17.** *$lv_\alpha(s,s') \leq ld_\alpha(s,s')$ for every $s, s' \in S$.*

The proof of Theorem 17 proceeds similarly to Lemma 2 in [9]. We will see, however, that this upper bound on lv_α is stronger (or at least no worse) than the bound obtained in [9]. Recall from [9] that bd_α is defined as the least fixed point of

$$\Gamma^{\Delta}_{\alpha}(d)(s,s') = \begin{cases} K^{\Delta}_{\alpha}(d)(\mu_s, \mu_{s'}) & \ell(s) = \ell(s') \\ 1 & \text{otherwise} \end{cases}$$

where $K^{\Delta}_{\alpha}(d)$ behaves as $K^{\Lambda}_{\alpha}(d)$, but uses $\Delta_\alpha(a,b) = \max\{a - \alpha b, b - \alpha a, 0\}$ rather than $\Lambda_\alpha(a,b) = \max\{a - \alpha b, 0\}$.

$$\begin{aligned}
\text{LD-}\textsc{threshold}(s,s',\theta) = \quad & \exists (d_{i,j})_{i,j\in S} \quad \bigwedge_{i,j\in S} (0 \le d_{i.j} \le 1) \quad \wedge \ d_{s,s'} \le \theta \\
& \wedge \ \bigwedge_{q,q'\in S} \begin{cases} d_{q,q'} = 1 & \ell(q) \ne \ell(q') \\ \mathit{couplingConstraint}(d,q,q') & \ell(q) = \ell(q') \end{cases}
\end{aligned}$$

$$\mathit{couplingConstraint}(d,q,q') = \exists (\omega_{i,j})_{i,j\in S} \quad \exists (\gamma_i)_{i\in S} \quad \exists (\tau_i)_{i\in S} \quad \exists (\eta_i)_{i\in S}$$

$$\sum_{i,j\in S} \omega_{i,j}\cdot d_{i,j} + \sum_i \eta_i \le d_{q,q'} \quad \wedge \quad \bigwedge_{i,j\in S} (0 \le \omega_{i,j} \le 1) \quad \wedge \quad \bigwedge_{i\in S} \begin{cases} 0 \le \gamma_i \le 1 \\ 0 \le \tau_i \le 1 \\ 0 \le \eta_i \le 1 \end{cases}$$

$$\wedge \bigwedge_{i\in S} (\sum_{j\in S} \omega_{i,j} - \gamma_i + \tau_i + \eta_i = \mu_q(i)) \quad \wedge \bigwedge_{j\in S} (\sum_{i\in S} \omega_{i,j} + \frac{\tau_j - \gamma_j}{\alpha} \le \mu_{q'}(j))$$

Figure 3 **NP** Formula for LD-THRESHOLD.

▶ **Theorem 18.** $\max\{ld_\alpha(s,s'), ld_\alpha(s',s)\} \le bd_\alpha(s,s')$ *for every* $s,s' \in S$.

Proof. Given a matrix A, let A^T be its transpose. Consider bd_α and ld_α as matrices. bd_α is the least fixed point of Γ^Δ_α so $\Gamma^\Delta_\alpha(bd_\alpha)(s,s') = bd_\alpha(s,s')$. Also notice that $\Gamma^\Lambda_\alpha(bd_\alpha)(s,s') \le \Gamma^\Delta_\alpha(bd_\alpha)(s,s')$, since $K^\Lambda_\alpha(bd_\alpha) \sqsubseteq K^\Delta_\alpha(bd_\alpha)$. To see this, note that, because $bd_\alpha = bd_\alpha^\mathsf{T}$, the relevant set of functions is the same, but the objective function in the supremum is smaller.

Hence $\Gamma^\Lambda_\alpha(bd_\alpha) \sqsubseteq bd_\alpha$, i.e. bd_α is also a pre-fixed point of Γ^Λ_α. Since ld_α is the least pre-fixed point of Γ^Λ_α then we know $ld_\alpha \sqsubseteq bd_\alpha$. By symmetry, $bd_\alpha = bd_\alpha^\mathsf{T}$ giving $ld_\alpha \sqsubseteq bd_\alpha^\mathsf{T}$ and then $ld_\alpha^\mathsf{T} \sqsubseteq bd_\alpha$. We conclude $\max\{ld_\alpha(s,s'), ld_\alpha(s',s)\} \le bd_\alpha(s,s')$ for every $s,s' \in S$. ◀

▶ Remark. Example 32 on page 13 demonstrates the inequality in Theorem 18 can be strict.

The standard variant of the Kantorovich metric is often presented in its dual formulation. In the case of finite distributions, the asymmetric skewed Kantorovich distance exhibits a dual form. This is obtained through the standard recipe for dualising linear programming. Interestingly, this technique yields a linear optimisation problem over a polytope independent of d, and that will prove useful in the computation of ld_α.

▶ **Lemma 19.** *Let X be finite and given $d : X\times X \to [0,1]$ a distance function, $\mu,\mu' \in \mathit{Dist}(X)$ we have*

$$K^\Lambda_\alpha(d)(\mu,\mu') = \min_{(\omega,\eta)\in\Omega^\alpha_{\mu,\mu'}} \Big(\sum_{s,s'\in X} \omega_{s,s'}\cdot d(s,s') + \sum_{s\in X} \eta_s \Big), \qquad \textit{where}$$

$$\Omega^\alpha_{\mu,\mu'} = \left\{ (\omega,\eta) \in [0,1]^{X\times X} \times [0,1]^X \quad \middle| \quad \begin{array}{l} \exists \gamma,\tau \in [0,1]^X \\ \forall i : \sum_j \omega_{i,j} + \tau_i - \gamma_i + \eta_i = \mu(i) \\ \forall j : \sum_i \omega_{i,j} + \frac{\tau_j - \gamma_j}{\alpha} \le \mu'(j) \end{array} \right\}.$$

When we refer to distance between states ($X = S$) we write $\Omega^\alpha_{s,s'}$ to mean $\Omega^\alpha_{\mu_s,\mu_{s'}}$. We take $V(\Omega^\alpha_{s,s'})$ to be the vertices of the polytope.

▶ **Theorem 20.** *ld_α can be computed in polynomial time with access to an* **NP** *oracle.*

We first show that the LD-THRESHOLD problem, which asks if $ld_\alpha(s,s') \le \theta$, is in **NP**. This is achieved through the formula shown in Figure 3, based on Lemma 19 and [30] which used a similar formula to approximate bisimilarity distances. The problem can be solved in **NP** as each of the variables can be shown to be satisfied in the optimal solution with rational numbers that are of polynomial size (see [9, Theorems 1 and 2]). It suffices to guess these numbers (non-deterministically) and verify the correctness of the formula in polynomial time.

Since the threshold problem can be solved in **NP**, we can approximate the value using binary search with polynomial overhead to arbitrary accuracy γ, thus we find a value x such that $|x - ld_\alpha(s,s')| \leq \gamma$. In fact, one can find the exact value of $ld_\alpha(s,s')$ in polynomial time assuming the oracle. We can show the value of ld_α is rational and its size is polynomially bounded, one can find it by approximation to a carefully chosen level of precision and then finding the relevant rational with the continued fraction algorithm [18, Section 5.1][16].

7 A greatest fixed point bound lgd_α

In the previous section we have used the least fixed point of Γ^Λ_α, which finds the fixed point closest to our objective lv_α. We now consider relaxing this requirement so that we can find a fixed point in polynomial time. We will introduce lgd_α, expressing the greatest fixed point and represent it as a linear program that can be solved in polynomial time. Relaxing to any fixed point could of course be much worse than ld_α, so we first refine our fixed point function (Γ^Λ_α) to reduce the potential gap. We do this by characterising the elements which are zero in ld_α and fixing these as such; so that they cannot be larger in the greatest fixed point.

Refinement of Γ^Λ_α

In the case of standard bisimulation distances the kernel of ld_1, that is $\{(s,s') \mid ld_1(s,s') = 0\}$, is exactly bisimilarity. We consider the kernel for ld_α and define a new relation $\sim_\alpha$, which we call skewed bisimilarity, which captures zero distance.

▶ **Definition 21.** *Let a relation $R \subseteq S \times S$ have the property*

$$(s,s') \in R \iff \exists\ (\omega,\eta) \in \Omega^\alpha_{s,s'}\ s.t.\ (\omega_{u,v} > 0 \implies (u,v) \in\ R) \quad \wedge \quad \forall u\ \eta_u = 0.$$

Arbitrary unions of such relations also maintain the property, thus a largest such relation exists. Let $\sim_\alpha$ be the largest relation with this property.

▶ Remark. When $\alpha = 1$ the formulation corresponds to an alternative characterisation of bisimilarity [19, 27], so $\sim_1 = \sim$.

▶ **Lemma 22.** *$ld_\alpha(s,s') = 0$ if and only if $s \sim_\alpha s'$.*

Since $ld_\alpha(s,s') = 0$ implies $lv_\alpha(s,s') = 0$, this also provides a way to show that δ is zero, that is, to show ϵ-differential privacy holds. However, note this is not a complete method to do this, and there are bisimilarity distances focused on finding ϵ [6].

▶ **Lemma 23.** *If $s \sim_\alpha s'$ then $lv_\alpha(s,s') = 0$.*

We need to be able to quickly and independently compute which pairs of states are related by $\sim_\alpha$. In fact we can do this in polynomial time using a closure procedure, which will terminate after polynomially many rounds.

▶ **Proposition 24.** *$\sim_\alpha$ can be computed in polynomial time in size$(\mathcal{M})$.*

Proof. We present a standard refinement algorithm, let $A_0 = S \times S$ and compute $A_{i+1} = \{(s,s') \in A_i \mid \exists (\omega,\eta) \in \Omega^\alpha_{s,s'} : \eta = \mathbf{0}\ \wedge\ (\omega_{u,v} > 0 \implies (u,v) \in A_i)\}$. To find this, define $\mathbb{1}_{!A_i}$, a matrix such that $\mathbb{1}_{!A_i}(s,s') = 0$ if $(s,s') \in A_i$ and 1 otherwise. Apply Γ^Λ_α to $\mathbb{1}_{!A_i}$, which amounts to computing n^2 linear programs. Take A_{i+1} to be indices of the matrix where $\Gamma^{\Lambda}_{\alpha}(\mathbb{1}_{!A_i})$ is zero. At each step, we remove at least one element, or stabilise so that the set will not change in subsequent rounds. After n^2 steps it is either stable or empty.

$A_{n^2} \subseteq \sim_\alpha$: after convergence we have some set such that $(s,s') \in A_{n^2} \implies \exists(\omega,\eta) \in \Omega^{\alpha}_{s,s'} : \eta = \mathbf{0} \, \wedge \, (\omega_{u,v} > 0 \implies (u,v) \in A_{n^2})$. $\sim_\alpha$ is the largest such set, so it contains A_{n^2}.

$\sim_\alpha \subseteq A_{n^2}$: by induction we start with $\sim_\alpha \subseteq A_0$ and only remove pairs not in $\sim_\alpha$. ◀

Recall that ld_α was defined as the least fixed point of $\Gamma^{\Lambda}_{\alpha}$. Let us refine $\Gamma^{\Lambda}_{\alpha}$ so the gap between the least fixed point and the greatest is as small as possible. We do this by fixing the known values of the least fixed point in the function, in particular the zero cases. We let

$$\Gamma'^{\Lambda}_{\alpha}(d)(s,s') = \begin{cases} 0 & s \sim_\alpha s' \\ \Gamma^{\Lambda}_{\alpha}(d)(s,s') & \text{otherwise} \end{cases}$$

and observe that ld_α is also the least fixed point of $\Gamma'^{\Lambda}_{\alpha}$.

▶ **Lemma 25.** *ld_α is the least fixed point of $\Gamma'^{\Lambda}_{\alpha}$.*

Definition and Computation of lgd_α

Towards a more efficiently computable function, we now study the greatest fixed point.

▶ **Definition 26.** *We let lgd_α be the greatest fixed point of $\Gamma'^{\Lambda}_{\alpha}$.*

It is equivalent to consider the greatest *post*-fixed point. It turns out that when $\alpha = 1$, $lgd_1 = ld_1$ [7]. We do not know if this holds for $\alpha > 1$, although conjecture that it might. Whilst it may not necessarily be as tight a bound on lv_α as ld_α, we can also use lgd_α to bound lv_α, thus the δ parameter of (ϵ,δ)-differential privacy. Because $ld_\alpha(s,s') \le lgd_\alpha(s,s')$ for every $s,s' \in S$, then Theorem 17 implies that $lv_\alpha(s,s') \le lgd_\alpha(s,s')$, for every $s,s' \in S$.

We will show that lgd_α can be computed in polynomial time using the ellipsoid method for solving a linear program of exponential size, matching the result of [7] for standard bisimilarity distances. Whilst we will not need to express the entire linear program in one go, we may need any one constraint at a time, so we need to be able to express each constraint, in polynomially many bits. We show that the representation of vertices of $\Omega^{\alpha}_{s,s'}$ is small.

▶ **Lemma 27.** *Each $(\omega,\eta) \in V(\Omega^{\alpha}_{s,s'})$ are rational numbers requiring a number of bits polynomial in size$(\mathcal{M})$.*

Proof. Consider the polytope:

$$\Omega'^{\alpha}_{\mu,\mu'} = \left\{ (\omega,\tau,\gamma,\eta) \in [0,1]^{S\times S} \times ([0,1]^S)^3 \quad \middle| \quad \begin{matrix} \forall i : \sum_j \omega_{i,j} + \tau_i - \gamma_i + \eta_i = \mu(i) \\ \forall j : \sum_i \omega_{i,j} + \frac{\tau_j - \gamma_j}{\alpha} \le \mu'(j) \end{matrix} \right\}$$

Each vertex is the intersection of hyperplanes defined in terms of μ,μ' (rationals given in the input $\mathcal{M}$), thus vertices of $\Omega'^{\alpha}_{\mu,\mu'}$ are rationals with representation size polynomial in the input. Vertices of $\Omega^{\alpha}_{\mu,\mu'} = \{(\omega,\eta) \mid \exists \tau,\gamma \; (\omega,\tau,\gamma,\eta) \in \Omega'^{\alpha}_{\mu,\mu'}\}$ require only fewer bits. ◀

The following linear program (LP) expresses the greatest post-fixed point. It has polynomially many variables but exponentially many constraints (for each s,s' one constraint for each $\omega \in V(\Omega^{\alpha}_{s,s'})$). Since linear programs can be solved in polynomial time, the greatest fixed point can be found in exponential time using the exponential size linear program.

▶ **Proposition 28.** *lgd_α is the optimal solution, $d \in [0,1]^{S\times S}$ of the following linear program: $\max_{d\in[0,1]^{S\times S}} \sum_{(u,v)\in S\times S} d_{u,v}$ subject to: for all $s, s' \in S$:*

$$\begin{aligned} d_{s,s'} &= 0 && && \textit{whenever } s \sim_\alpha s', \\ d_{s,s'} &= 1 && && \textit{whenever } \ell(s) \neq \ell(s'), \\ d_{s,s'} &\leq \sum_{(u,v)\in S\times S} \omega_{u,v} d_{u,v} + \sum_{u\in S} \eta_u && \textit{for all } (\omega,\eta) \in V(\Omega^\alpha_{s,s'}) && \textit{otherwise.} \end{aligned}$$

Proof. The $s \sim_\alpha s'$ and $\ell(s) \neq \ell(s')$ cases follow by definition. Observe that by the definition of lgd_α as a post-fixed point it is required that $d(s,s') \leq \Gamma'^{\Lambda}_{\alpha}(d)(s,s') = K^{\Lambda}_{\alpha}(d)(s,s') = \min_{(\omega,\eta)\in\Omega^\alpha_{s,s'}} \sum_{(u,v)\in S\times S} \omega_{u,v} d_{u,v} + \sum_{u\in S} \eta_u$ or equivalently, for all $(\omega,\eta) \in \Omega^\alpha_{s,s'}$: $d(s,s') \leq \sum_{(u,v)\in S\times S} \omega_{u,v} d_{u,v} + \sum_{u\in S} \eta_u$ ◀

In the spirit of [7], we can solve the exponential-size linear program given in Proposition 28 using the ellipsoid method, in polynomial time. Whilst the linear program has exponentially many constraints, it has only polynomially many variables. Therefore, the ellipsoid method can be used to solve the linear program in polynomial time, provided a polynomial-time separation oracle can be given [26, Chapter 14]. Separation oracle takes as argument $d \in [0,1]^{S\times S}$, a proposed solution to the linear program and must decide whether d satisfies the constraints or not. If not then it must provide $\theta \in \mathbb{Q}^{|S\times S|}$ as a separating hyperplane such that, for every d' that does satisfy the constraints, $\sum_{u,v} d_{u,v}\theta_{u,v} < \sum_{u,v} d'_{u,v}\theta_{u,v}$.

Our separation oracle will perform the following: for every $s, s' \in S$ check that $d(s,s') \leq \min_{(\omega,\eta)\in\Omega^\alpha_{s,s'}} \omega \cdot d + \eta \cdot \mathbf{1}$. This is done by solving $\min_{(\omega,\eta)\in\Omega^\alpha_{s,s'}} \omega \cdot d + \eta \cdot \mathbf{1}$ using linear programming. If every check succeeds, return YES. If some check fails for s, s' return NO and

$$\theta_{u,v} = \begin{cases} \omega_{u,v} - 1 & (u,v) = (s,s') \\ \omega_{u,v} & \text{otherwise} \end{cases} \qquad \text{where } (\omega,\eta) = \operatorname*{argmin}_{(\omega,\eta)\in V(\Omega^\alpha_{s,s'})} d\cdot\omega + \eta\cdot\mathbf{1}.$$

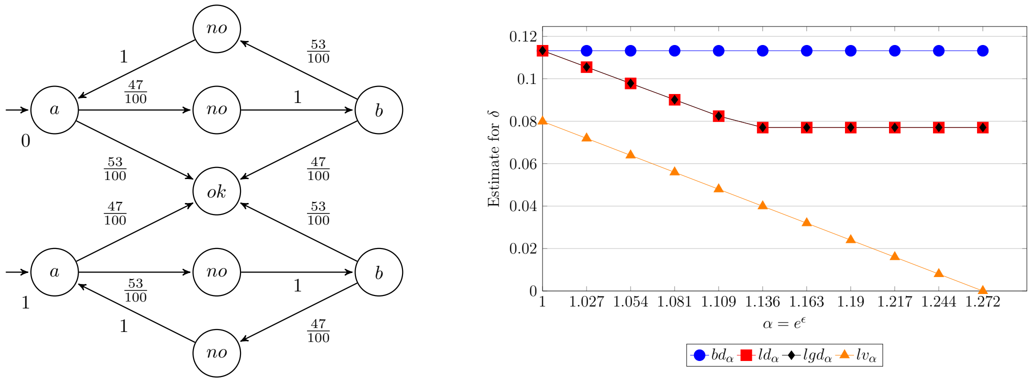
▶ **Lemma 29.** *θ is a separating hyperplane, i.e., it separates the unsatisfying d and all satisfying d'.*

▶ **Theorem 30.** *lgd_α can be found in polynomial time in the size of $\mathcal{M}$.*

Proof. Checking $d(s,s') \leq \min_{\omega,\eta\in\Omega^\alpha_{s,s'}} \omega \cdot d + \eta \cdot \mathbf{1}$ is polynomial time. The linear program is of polynomial size, so runs in polynomial time in the size of the encoding of the linear program. Similarly finding θ is polynomial time by running essentially the same linear program and reading off the minimising result.

Because pairs (ω,η) are in $V(\Omega^\alpha_{s,s'})$, they are polynomial size in the size of $\mathcal{M}$, independent of d, by Lemma 27. Note that, unlike in Chen et al. [7], the oracle procedure is not strongly polynomial, so the time to find θ may depend on the size of d, but the output θ and d remain polynomial in the size of the initial system.

We conclude there is a procedure for computing lgd_α running in polynomial time [26, Theorem 14.1, Page 173]. There exists a polynomial ψ where the ellipsoid algorithm solves the linear program in time $T \cdot \psi(\mathit{size}(\mathcal{M}))$, where T is the time the separation algorithm takes on inputs of size $\psi(\mathit{size}(\mathcal{M}))$. Since the $T \in \mathit{poly}(\psi(\mathit{size}(\mathcal{M})))$ and $\psi(\mathit{size}(\mathcal{M})) \in \mathit{poly}(\mathit{size}(\mathcal{M}))$ then $T \in \mathit{poly}(\mathit{size}(\mathcal{M}))$. Overall we have $T \cdot \psi(\mathit{size}(\mathcal{M})) \in \mathit{poly}(\mathit{size}(\mathcal{M}))$. ◀



(a) Labelled Markov chain.

(b) Calculated approximations of δ given ϵ.

Figure 4 PIN Checker example: each state denotes its label, transition probabilities on arrows.

8 Examples

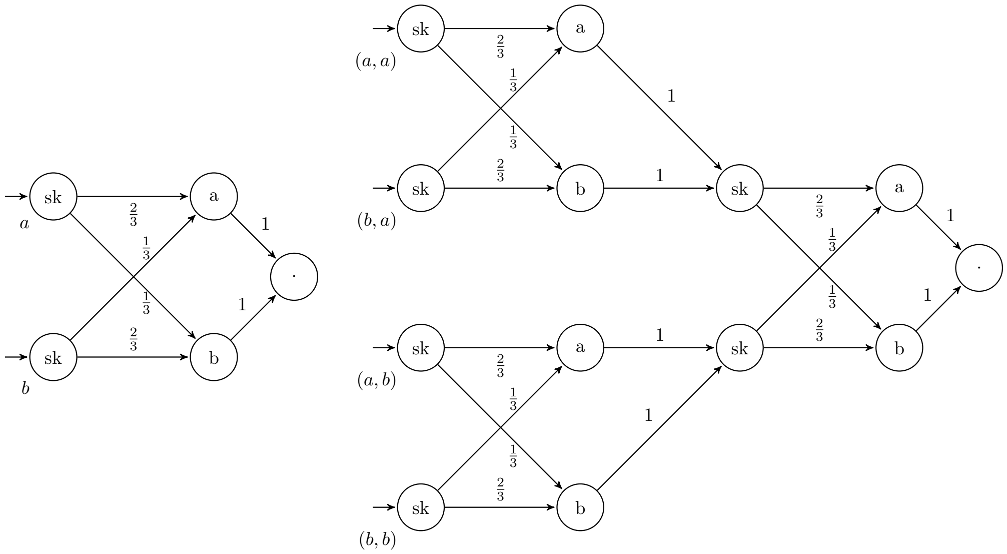
▶ **Example 31** (PIN Checker). We demonstrate our methods are a sound technique for determining the δ privacy parameter (given e^{ϵ}, where ϵ is the other privacy parameter). We take as an example, in Figure 4, a PIN checking system from [32, 31]. Intuitively, the machine accepts or rejects a code (a or b). Instead of accepting a code deterministically, it probabilistically decides whether to accept. The machine allows an attempt with the other code if it is not accepted. We model the system that accepts more often on the the pin-code a, from state 0, and the system that accepts more often from code b, from state 1. The chain simulates attempts to gain access to the system by trying code a then b until the system accepts (reaching the "end" state). Pen-and-paper analysis can determine that the system is $(\ln(\frac{2809}{2209}), 0)$-differentially private, or at the other extreme $(0, \frac{200}{2503})$-differentially private ($\frac{2809}{2209} \approx 1.27$, $\frac{200}{2503} \approx 0.0799$). The true privacy, lv_α is shown along the orange line (▲).

In the blue line (●) we see the estimate bd_α as defined in [9]; which correctly bounds the true privacy, but is unresponsive to α. Using the methods introduced in this paper we compute ld_α on the red line (■) and lgd_α on the black line (◆), which coincide. We observe that this is an improvement and is within approximately 1.5 times the true privacy for $\alpha \leq 1.035$. In this example observe that $ld_\alpha = lgd_\alpha$; suggesting lgd_α, which can be computed in polynomial time is as good as ld_α. Our results do eventually suffer, as increasing α cannot find a better δ, despite a lower value existing.

▶ **Example 32** (Randomised Response). The randomised response mechanism allows a data subject to reveal a secret answer to a potentially humiliating or sensitive question honestly with some degree of plausible deniability. This is achieved by flipping a biased coin and providing the wrong answer with some probability based on the coin toss. If there are two answers a or b, answering truthfully with probability $\frac{\beta}{1+\beta}$ and otherwise with $\frac{1}{1+\beta}$ leads to ϵ-differential privacy where $e^{\epsilon} = \beta$ and such a bound is tight (there is no smaller ϵ' such that answering in this way gives ϵ'-differential privacy). However, it can be (ϵ', δ)-differentially private for $\epsilon' < \epsilon$ and some δ.

Let us consider the single-input, single-output randomised response mechanism shown in Figure 5a with $\beta = 2$, hence $\ln(2)$-differentially private, alternatively it is $(\ln(\frac{6}{5}), \frac{4}{15})$-differential privacy ($\ln(\frac{6}{5}) \approx \frac{\ln(2)}{4}$). We consider the application of composing automata to determine more complex properties automatically.

Differential privacy enjoys multiple composition theorems [15]. When applied to disjoint datasets, differential privacy allows the results of (ϵ, δ)-differentially private mechanism applied to each independently to be combined with no additional loss in privacy. Let us consider the



(a) Single-input, single-output. **(b)** Two-input, two-output.

Figure 5 Randomised response. Every second label is the outcome of the randomised response mechanism and alternately `sk` (for "skip"). The left most state represents the sensitive input.

two-input, two-output labelled Markov chain (Figure 5b), where we consider each input to be from two independent respondents, using our methods verifies that the privacy does not increase on the partitioned data. We consider the adjacency relation as the symmetric closure of $R = \{((a,a),(a,b)),((a,a),(b,a)),((b,b),(a,b)),((b,b),(b,a))\}$. We determine $(\ln(\frac{6}{5}), \frac{4}{15})$-differential privacy by computing $\max_{(s,s')\in R} ld_{6/5}(s,s') = \frac{4}{15}$, verifying there is no privacy loss from composition. Because randomised response is finite we can compute lv_α for adjacent inputs in exponential time for comparison. In this instance, our technique provides the optimal solution, in the sense $\max_{(s,s')\in R} ld_{6/5}(s,s') = \max_{(s,s')\in R} lv_{6/5}(s,s')$; indicating that ld_α and lgd_α can provide a good approximation.

The basic composition theorems suggest that if a mechanism that is (ϵ,δ)-differentially private is used k times, one achieves $(k\epsilon, k\delta)$-differential privacy [13]. However, this is not necessarily optimal. More advanced composition theorems may enable tighter analysis, although this can can be computationally difficult (#**P**-complete) [25]. Even this may not be exact when allowed to look inside the composed mechanisms. If we assume the responses are from two questions answered by the same respondent and let $R' = R \cup \{((a,a),(b,b))\}$, naively applying basic composition concludes $(\ln(\frac{36}{25}), \frac{8}{15})$-differential privacy. Our methods can find a better bound than basic composition since $\max_{(s,s')\in R'} ld_{36/25}(s,s') = \frac{103}{225} < \frac{8}{15}$. However, in this case, our technique is not optimal either.

9 Conclusion

Our results are summarised in Figure 1 on page 2. We are interested in the value of lv_α, but it is not computable and difficult to approximate. We have defined an upper bound ld_α, showing that it is more accurate than the previously known bound bd_α from [9] and just as easy to compute (in polynomial time with an **NP** oracle). We also defined a distance based on the greatest fixed point, lgd_α, which has the same flavour but can be computed in polynomial time. When considering lv_α directly, we approximate to arbitrary precision in **PSPACE** and show it is #**P**-hard (which generalises a known result on tv). It is open whether the least fixed point bisimilarity distance (or any refinement smaller than lgd_α) can be computed in polynomial time, or even if $lgd_\alpha = ld_\alpha$. It is also open whether approximation can be resolved to be in #**P**, **PSPACE**-hard, or complete for some intermediate class.

References

1 Giorgio Bacci, Giovanni Bacci, Kim G. Larsen, and Radu Mardare. On-the-Fly Exact Computation of Bisimilarity Distances. 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 1–15. Springer, 2013. doi:10.1007/978-3-642-36742-7_1.

2 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

3 Gilles Barthe, Boris Köpf, Federico Olmedo, and Santiago Zanella Béguelin. Probabilistic relational reasoning for differential privacy. In John Field and Michael Hicks, editors, *Proceedings of the 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2012, Philadelphia, Pennsylvania, USA, January 22-28, 2012*, pages 97–110. ACM, 2012. doi:10.1145/2103656.2103670.

4 Gilles Barthe and Federico Olmedo. Beyond Differential Privacy: Composition Theorems and Relational Logic for f-divergences between Probabilistic Programs. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 49–60. Springer, 2013. doi:10.1007/978-3-642-39212-2_8.

5 Patrick Billingsley. *Probability and Measure.* John Wiley and Sons, 2nd edition, 1986.

6 Konstantinos Chatzikokolakis, Daniel Gebler, Catuscia Palamidessi, and Lili Xu. Generalized Bisimulation Metrics. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014 - Concurrency Theory - 25th International Conference, CONCUR 2014, Rome, Italy, September 2-5, 2014. Proceedings*, volume 8704 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2014. doi:10.1007/978-3-662-44584-6_4.

7 Di Chen, Franck van Breugel, and James Worrell. On the Complexity of Computing Probabilistic Bisimilarity. In Lars Birkedal, editor, *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, volume 7213 of *Lecture Notes in Computer Science*, pages 437–451. Springer, 2012. doi:10.1007/978-3-642-28729-9_29.

8 Taolue Chen and Stefan Kiefer. On the total variation distance of labelled Markov chains. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 33:1–33:10. ACM, 2014. doi:10.1145/2603088.2603099.

9 Dmitry Chistikov, Andrzej S. Murawski, and David Purser. Bisimilarity Distances for Approximate Differential Privacy. In Shuvendu K. Lahiri and Chao Wang, editors, *Automated Technology for Verification and Analysis - 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings*, volume 11138 of *Lecture Notes in Computer Science*, pages 194–210. Springer, 2018. Full version with proofs can be found at arXiv:1807.10015. doi:10.1007/978-3-030-01090-4_12.

10 Yuxin Deng and Wenjie Du. The Kantorovich Metric in Computer Science: A Brief Survey. *Electr. Notes Theor. Comput. Sci.*, 253(3):73–82, 2009. doi:10.1016/j.entcs.2009.10.006.

11 Josee Desharnais, Vineet Gupta, Radha Jagadeesan, and Prakash Panangaden. Metrics for labelled Markov processes. *Theor. Comput. Sci.*, 318(3):323–354, 2004. doi:10.1016/j.tcs.2003.09.013.

12 Josee Desharnais, Radha Jagadeesan, Vineet Gupta, and Prakash Panangaden. The Metric Analogue of Weak Bisimulation for Probabilistic Processes. In *17th IEEE Symposium on Logic in Computer Science (LICS 2002), 22-25 July 2002, Copenhagen, Denmark, Proceedings*, pages 413–422. IEEE Computer Society, 2002. doi:10.1109/LICS.2002.1029849.

13 Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our Data, Ourselves: Privacy Via Distributed Noise Generation. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006, 25th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28 - June 1, 2006, Proceedings*, volume 4004 of *Lecture Notes in Computer Science*, pages 486–503. Springer, 2006. doi:10.1007/11761679_29.

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

15 Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014. doi:10.1561/0400000042.

16 Kousha Etessami and Mihalis Yannakakis. On the Complexity of Nash Equilibria and Other Fixed Points. *SIAM J. Comput.*, 39(6):2531–2597, 2010. doi:10.1137/080720826.

17 Alessandro Giacalone, Chi-Chang Jou, and Scott A. Smolka. Algebraic Reasoning for Probabilistic Concurrent Systems. In Manfred Broy, editor, *Programming concepts and methods: Proceedings of the IFIP Working Group 2.2, 2.3 Working Conference on Programming Concepts and Methods, Sea of Galilee, Israel, 2-5 April, 1990*, pages 443–458. North-Holland, 1990.

18 Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric Algorithms and Combinatorial Optimization*, volume 2 of *Algorithms and Combinatorics*. Springer, 1988. doi:10.1007/978-3-642-97881-4.

19 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.

20 Sampath Kannan, Z Sweedyk, and Steve Mahaney. Counting and random generation of strings in regular languages. In *Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 551–557. Society for Industrial and Applied Mathematics, 1995.

21 L. V. Kantorovich. On the translocation of masses. *Doklady Akademii Nauk SSSR*, 37(7-8):227—-229, 1942.

22 Stefan Kiefer. On Computing the Total Variation Distance of Hidden Markov Models. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 130:1–130:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. Full version with proofs can be found at arXiv:1804.06170. doi:10.4230/LIPIcs.ICALP.2018.130.

23 Kim Guldstrand Larsen and Arne Skou. Bisimulation through Probabilistic Testing. *Inf. Comput.*, 94(1):1–28, 1991. doi:10.1016/0890-5401(91)90030-6.

24 Sebastian Meiser. Approximate and Probabilistic Differential Privacy Definitions. *IACR Cryptology ePrint Archive*, 2018:277, 2018. URL: https://eprint.iacr.org/2018/277.

25 Jack Murtagh and Salil P. Vadhan. The Complexity of Computing the Optimal Composition of Differential Privacy. In Eyal Kushilevitz and Tal Malkin, editors, *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings,*

Part I, volume 9562 of *Lecture Notes in Computer Science*, pages 157–175. Springer, 2016. doi:10.1007/978-3-662-49096-9_7.

26 Alexander Schrijver. *Theory of linear and integer programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

27 Qiyi Tang and Franck van Breugel. Computing Probabilistic Bisimilarity Distances via Policy Iteration. 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 22:1–22:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016. doi:10.4230/LIPIcs.CONCUR.2016.22.

28 Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

29 Franck van Breugel. Probabilistic bisimilarity distances. *SIGLOG News*, 4(4):33–51, 2017. URL: https://dl.acm.org/citation.cfm?id=3157837.

30 Franck van Breugel, Babita Sharma, and James Worrell. Approximating a Behavioural Pseudometric Without Discount for Probabilistic Systems. 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 123–137. Springer, 2007. doi:10.1007/978-3-540-71389-0_10.

31 Lili Xu. *Formal Verification of Differential Privacy in Concurrent Systems*. PhD thesis, Ecole Polytechnique (Palaiseau, France), 2015.

32 Lili Xu, Konstantinos Chatzikokolakis, and Huimin Lin. Metrics for Differential Privacy in Concurrent Systems. In Erika Ábrahám and Catuscia Palamidessi, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 34th IFIP WG 6.1 International Conference, FORTE 2014, Held as Part of the 9th International Federated Conference on Distributed Computing Techniques, DisCoTec 2014, Berlin, Germany, June 3-5, 2014. Proceedings*, volume 8461 of *Lecture Notes in Computer Science*, pages 199–215. Springer, 2014. doi:10.1007/978-3-662-43613-4_13.

Event Structures for Mixed Choice

Marc de Visme
Univ Lyon, EnsL, UCBL, CNRS, LIP, F-69342, LYON Cedex 07, France

Abstract

In the context of models with mixed nondeterministic and probabilistic choice, we present a concurrent model based on partial orders, more precisely Winskel's event structures. We study its relationship with the interleaving-based model of Segala's probabilistic automata. Lastly, we use this model to give a truly concurrent semantics to an extension of CCS with probabilistic choice, and relate this concurrent semantics to the usual interleaving semantics, thus generalising existing results on CCS, event structures and labelled transition systems.

2012 ACM Subject Classification Theory of computation → Parallel computing models

Keywords and phrases probability, nondeterminism, concurrency, event structures, CCS

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.11

Related Version Full version at `https://hal.archives-ouvertes.fr/hal-02167420`.

Funding *Marc de Visme*: Supported by Labex MiLyon (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007), operated by the French National Research Agency (ANR).

1 Introduction

We study models of *mixed choice* [7], i.e. models representing both probabilistic choice and nondeterministic choice. The need for such models arises with systems where unconstrained nondeterministic behaviours coexist with quantified and controlled nondeterministic behaviours; for example, parallel threads using random number generators (hence probabilistic choices) while operating on a shared memory (hence nondeterministic races).

While many different models have been developed through the years, Segala's probabilistic automata [13, 14] are a widely used model, both general and practical. They are automata where transitions are from states to probability distributions on states, hence modelling an alternation between a nondeterministic choice (the choice of the transition) and a probabilistic choice (the probability distribution). It is an *interleaving model*, as it represents "A and B occur in parallel" by "A then B or B then A". In this paper, we are interested in models that are not interleaving, and represent "A and B occur in parallel" by "A and B are causally unrelated" instead. Those models are called *truly concurrent models*, and are particularly useful to study races, concurrency, and causality. We specifically focus on truly concurrent models based on partial orders, more precisely Winskel's event structures [9, 11, 17]. We present here *mixed probabilisitic event structures*, which are event structures enriched to model mixed nondeterministic and probabilistic choice. This work is in continuation of works on event structure models for languages with effects: parallelism [17], probabilities [5, 16], quantum effects [6], shared weak memory [4], . . .

In order to ensure that mixed probabilistic event structures are an adequate model for mixed choice, we show how to relate them to the existing model of Segala automata. Indeed, a mixed probabilistic event structure can be unfolded to a (tree-like) Segala automaton through a *sequentialisation* procedure, similar to the unfolding of a partial order into a tree. This sequentialisation procedure is well-behaved; rather than listing all the ad-hoc properties it satisfies, we express mixed probabilisitic event structures as a category, such that sequentialisation forms an adjunction, from which those properties can be deduced. Sections 3 and 4 are dedicated to the development of this new model for mixed choice.

We apply this model to give a semantics to a language featuring parallelism, nondeterminism and probabilistic choice. Namely, we choose an extension of CCS [10] with probabilistic choice [2]. Extensions of our work to more complex languages, such as the probabilistic π-calculus, should be possible using methods similar to [16]. In that paper, Varacca and Yoshida give a concurrent semantics using event structures to a "deterministic" probabilistic π-calculus, i.e. they forbid processes with a nondeterministic behaviour, like $(a|a|\bar{a})$ where two inputs race to interact with an output. We do not have those restrictions, and fully support both nondeterministic and probabilistic behaviours.

Using partial-order based models to represent parallel processes is not a new idea. Indeed, the relationship between CCS and event structures is well-studied [17], and recalled in detail in Section 2. The core result of this relationship is the factorisation theorem (Theorem 9), which states that if one considers the event structure representing the concurrent semantics of a process, and unfolds it into a labelled tree, then the result would match the interleaving semantics of the process. It follows that the concurrent semantics is *sound* w.r.t. the interleaving semantics. In Section 4, we lift the known relations between CCS, event structures, and labelled trees, to relations between Probabilistic CCS, mixed probabilisitic event structures, and (tree-like) Segala automata; concluding with a factorisation theorem (Theorem 26).

2 Preliminaries

The process algebra CCS [10] is used to represent concurrent processes communicating with each other through channels. A process P can evolve into a process Q by performing an action. The graph having processes as nodes and transitions labelled by actions as edges is known as the Labelled Transition System (LTS) of the process language. From this graph and any process P, one can work with the (possibly infinite) labelled tree obtained by unfolding the graph starting by the node P. We may express the semantics of a process P in terms of that unfolded tree, the approach we take in this paper.

The LTS of processes yield an *interleaving semantics*, since they do not distinguish the process $(a|b)$ that perform a and b in parallel from the process $(a.b \varovee b.a)$ – where $\varovee$ represents nondeterministic choice – which can perform a and b in any sequential order. Semantics that do distinguish between the two actions in parallel and the two actions in any sequential order are called *truly concurrent semantics*. In this paper, we will consider the usual truly concurrent semantics of CCS: event structures [9, 11, 17], which are partial orders with a notion of conflict (Definition 1).

The semantics of CCS in labelled trees and in event structures are both quite straightforward, except for the parallel composition $(P|Q)$, which is syntactically complex to compute. It is often practical to use a more abstract approach than syntactically computing $(P|Q)$, and remark that the parallel composition of CCS arises in both semantics [17] as a categorical product $\times_\star$ (for a suitable notion of maps) followed by a restriction.

As stated before, we can recover the interleaving semantics from the truly concurrent semantics. This means that we can unfold event structures as labelled trees, while preserving the interpretation of CCS processes. This unfolding consists in finding the best labelled tree to approximate the behaviour of a given event structure.

In the literature of models of concurrency, following Winskel [12, 17], such unfoldings are often expressed through a categorical *coreflection* – a special case of adjunction. The unfolding of an event structure into a labelled tree, named *sequentialisation*, is the right adjoint of this adjunction, ensuring that the sequentialisation $\mathit{Seq}(E)$ of an event structure E gives its best approximation in terms of labelled trees. In particular the sequentialisation of a product $\mathit{Seq}(E \times_\star F)$ is the product of the sequentialisations $\mathit{Seq}(E) \times_\star \mathit{Seq}(F)$; hence the sequentialisation preserves the interpretation of the parallel composition. This adjunction is a *coreflection*, meaning that the left adjoint, named *inclusion*, is such that the composition $(\mathit{Seq} \circ \hookrightarrow)$ is an isomorphism – in other words, labelled trees can be regarded as a particular case of event structures.

In this section, we first recall the definition of event structures and their cartesian category. Then, we recall the syntax of the process algebra CCS, its interleaving semantics using labelled trees, and its truly concurrent semantics using event structures. Finally, we present the extension of CCS with probabilistic choice [2], and its interleaving semantics.

2.1 The Category of Event Structures

A (prime) event structure [18] is a representation of computational events of a concurrent system. Its events are related by a partial order representing causal dependency: if $a \leq b$ then b can only occur if a has occurred beforehand; and a conflict relation representing incompatibility: if $a \# b$ then each of a and b can only occur if the other does not.

▶ **Definition 1.** *An* event structure *E is a triple $E = (|E|, \leq_E, \#_E)$ where:*

- *$|E|$ is a set whose elements are called events.*
- *$\leq_E$ is a (partial) order, called causality, such that $\{b \mid b \leq_E a\}$ is finite for all $a \in |E|$.*
- *$\#_E$ is a symmetric irreflexive binary[1] relation, called conflict.*
- *$\forall a, b, c \in |E|$, if $a \#_E b \leq_E c$ then $a \#_E c$.*

In such a structure, we want to describe the set of states in which the system under study can exist. We define the set of (finite) configurations of E, written $\mathcal{C}(E)$, as the set of reachable finite sets of events, in other words:

$$x \in \mathcal{C}(E) \iff x \in \mathcal{P}_{\text{fin}}(|E|) \text{ and } \forall a \in x, \forall b \in |E|, \begin{cases} b \#_E a \implies b \notin x & \text{and} \\ b \leq_E a \implies b \in x \end{cases}$$

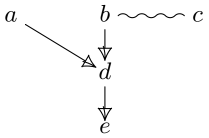
We say that a configuration x *enables* an event $a \notin x$, and we write $x \overset{a}{\longrightarrow\!\!\!\subset}$, if $x \cup \{a\}$ is a configuration. There are two canonical configurations associated to an event $a \in |E|$: the minimal configuration containing it $[a] := \{b \mid b \leq_E a\}$, and the minimal configuration enabling it $[a) := [a] \backslash \{a\}$.

The causality $\leq_E$ and the conflict $\#_E$ contain a lot of redundant information: if you know that $a \#_E b$ and $b \leq_E c$, then the definition of event structures enforces $a \#_E c$. When representing event structures, we want a concise representation, so instead of $\leq_E$ and $\#_E$, we use immediate causality $\rightarrowtriangle_E$ (represented by arrows) and minimal conflict $\sim\!\sim_E$ (represented by wiggly lines), defined as follows:

$$a \rightarrowtriangle_E b \iff \begin{cases} a <_E b \\ \forall a \leq_E c \leq_E b, c \in \{a, b\} \end{cases} \qquad a \sim\!\sim_E b \iff \begin{cases} a \#_E b \\ \forall a' <_E a, \neg(a' \#_E b) \\ \forall b' <_E b, \neg(a \#_E b') \end{cases}$$

We can always recover $\#_E$ and $\leq_E$ from $\rightarrowtriangle_E$ and $\sim\!\sim_E$. For example, in Figure 1, we deduce $c \#_E e$ and $a \leq_E e$ from the minimal conflict and immediate causality.

[1] We choose to use binary conflicts for pedagogical reasons. Our work can be extended to non-binary conflicts as in [19].



$$\mathcal{C}(E) = \left\{ \begin{array}{c} \varnothing, \{a\}, \{b\}, \{c\}, \{a,b\}, \\ \{a,c\}, \{a,b,d\}, \{a,b,d,e\} \end{array} \right\}$$

Figure 1 An event structure E.

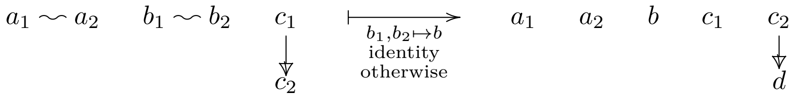


Figure 2 A total map of event structures.

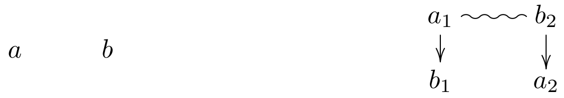


Figure 3 Concurrent system, and its interleaving counterpart.

There is a notion of (partial or total) maps of event structures, altogether forming a category. Formally, a (partial or total) map f from E to E' is a (partial or total) function $f : |E| \rightharpoonup |E'|$ such that:

configuration preserving for all $x \in \mathcal{C}(E)$ we have $f(x) \in \mathcal{C}(E')$.

local injectivity for all a, b distinct in $x \in \mathcal{C}(E)$, if f is defined on both then $f(a) \neq f(b)$.

Note that total maps can be interpreted as a form of simulation: if $x \in \mathcal{C}(E)$ enables an event $e \notin x$, then $f(x) \in \mathcal{C}(E')$ enables $f(e) \notin f(x)$. In the example of Figure 2, the map is total. It merges b_1 and b_2 into a single event b, which is allowed since they are in conflict, and it does not contain the event d in its image, which is allowed because d is not causally required by any event in the image of the map.

With this notion of maps, we recall the induced notion of isomorphism: $E \cong E'$ if and only if there is a total bijective map $f : E \to E'$ with f^{-1} also a map of event structures. In other words, two event structures are isomorphic if and only if they are the same up to renaming of the events.

▶ **Proposition 2.** *Event structures and (partial) maps of event structures form a category, written* $\mathsf{ES}_\star$*. We write* $\varnothing$ *for the empty event structure, which is a terminal object*[2]*.* $\mathsf{ES}_\star$ *has a subcategory* ES *of event structures with total maps of event structures.*

Event structures are used to represent causality and independence, but they can also be used to represent interleavings. However, concurrent systems and sequential systems simulating concurrency through interleaving will be represented in drastically different ways in event structures. Figure 3 shows a concurrent system able to perform two events a and b in parallel, and its interleaving counterpart where a and b can both happen in any order but not simultaneously. We can characterise event structures that are fully interleaved, in other words sequential event structures, as follows:

[2] i.e. for every object A, there exists a unique map from A to $\varnothing$.

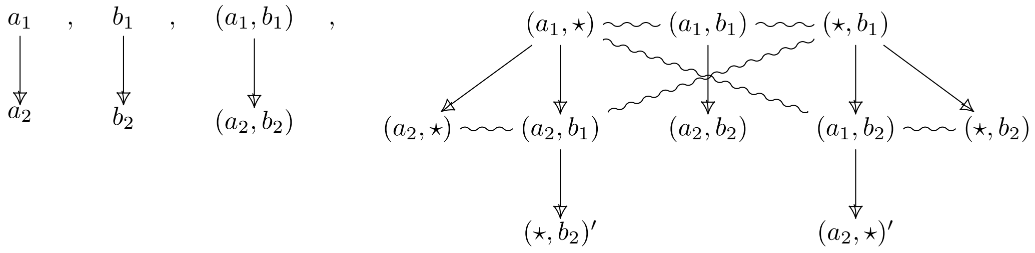


Figure 4 The event structure A, B, $A \times B$ and $A \times_\star B$.

▶ **Definition 3.** *An event structure E is sequential if it satisfies one of the following equivalent properties:*

- *E is forest-shaped, with branches being in conflict with each other.*
- *For every $a, b \in x \in \mathcal{C}(E)$, either $a \leq_E b$ or $b \leq_E a$.*
- *There exists a (necessarily unique) total map from E to $\mathcal{N}$, where $\mathcal{N} = (\mathbb{N}, \leq, \varnothing)$ is the event structure with an infinite succession of events.*
- *E is isomorphic to $E \times \mathcal{N}$, where $\times$ is the synchronous product defined in Section 2.2.*

The first characterisation means that trees are in a one-to-one correspondence with sequential event structures. Through this correspondence, transitions correspond to events. For the remainder of this paper, we use sequential event structures to represent trees.

▶ **Proposition 4.** *Sequential event structures and (partial) maps of event structures form a subcategory of* $\mathsf{ES}_\star$*, written* $\mathsf{Seq\text{-}ES}_\star$*. It has a subcategory* $\mathsf{Seq\text{-}ES}$ *of sequential event structures with total maps of event structures, with $\mathcal{N}$ for terminal object.*

2.2 The Categorical Product

In this paper, we exclusively rely on the universal property of categorical products, without giving any concrete definition for such products. We write $\times_\star$ for the product in $\mathsf{ES}_\star$ [17], called *asynchronous product*, and $\times$ for the product in ES [17], called *synchronous product*.

Informally speaking, both $A \times_\star B$ and $A \times B$ explore all the ways to pair up the events of A with the events of B while respecting causal dependencies and conflicts. The synchronous product expects all the events to be paired, while the asynchronous product allows events to remain unpaired, see Figure 4. Note that events of the products are not uniquely determined by their projections: in Figure 4 both $(\star, b_2)$ and $(\star, b_2)'$ have the same projections.

▶ **Proposition 5.** $\mathsf{ES}_\star$ *is a cartesian category, with $\times_\star$ as a product and $\varnothing$ as a unit.* ES *has a product $\times$, and its subcategory* $\mathsf{Seq\text{-}ES}$ *is a cartesian category with $\mathcal{N}$ as unit.*

Using the synchronous product $\times$, we can simply define the sequentialisation as $Seq(E) := E \times \mathcal{N}$. It is always a sequential event structure. The asynchronous product $\times_\star$ will later be used to define the semantics of the parallel composition of CCS.

▶ **Theorem 6.** *Seq is a functor from* $\mathsf{ES}_\star$ *(and* ES*) to* $\mathsf{Seq\text{-}ES}_\star$ *(and* $\mathsf{Seq\text{-}ES}$*), which is right adjoint to the inclusion functor, forming coreflections [17].*

$$\begin{tikzcd}
\mathsf{Seq\text{-}ES} \arrow[r, bend left, "\hookrightarrow"] \arrow[r, phantom, "\perp"] & \mathsf{ES} \arrow[l, bend left, "Seq"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{Seq\text{-}ES}_\star \arrow[r, bend left, "\hookrightarrow"] \arrow[r, phantom, "\perp"] & \mathsf{ES}_\star \arrow[l, bend left, "Seq"]
\end{tikzcd}$$

Since right adjoints preserve categorical limits, we have that $\mathit{Seq}(A \times_\star B) \cong \mathit{Seq}(A) \times_\star \mathit{Seq}(B)$. Since the restriction (Definition 8) is also preserved by Seq, it means that when interpreting the parallel composition of CCS using the asynchronous product in $\mathsf{ES}_\star$, then sequentialising, we get the same result as when interpreting the parallel composition directly as the asynchronous product in $\mathsf{Seq\text{-}ES}_\star$ (i.e. trees).

2.3 Semantics of CCS

We consider the process algebra CCS [1, 3, 10] over a set of channels *Chan*. A channel $a \in \mathit{Chan}$ can be used by processes as an input a or as an output $\bar{a}$. For convenience, we assume that $\bar{\bar{a}} = a$. On top of these external actions, processes can also perform internal actions, written τ, and we write $\mathbb{A} = \mathit{Chan} \cup \{\bar{a} \mid a \in \mathit{Chan}\} \cup \{\tau\}$ for the set of all actions.

▶ **Definition 7.** *The set $\mathbb{P}$ of processes is defined by the following syntax:*

$P ::= \mathbf{0} \mid (P_1|P_2) \mid (P_1 \ovee P_2) \mid X \mid \mu X.P \mid a.P \mid \bar{a}.P \mid \tau.P \mid \nu a.P$ *(for $a \in \mathit{Chan}$)*

where the constructs are empty process, parallel composition, nondeterministic choice, process variable, process recursion, input prefix, output prefix and τ prefix, and restriction, respectively. We only consider closed processes, in which every variable X is bound by a recursion μX.

Figure 5a describes the interleaving semantics of the language. The states of the LTS reachable from a process P can be unfolded into a (potentially infinite) labelled tree represented using a (potentially infinite) sequential event structure, written $[\![P]\!]_{\mathrm{is}}$, with labels in $\mathbb{A}$.

▶ **Definition 8.** *An event structure with labels in $\mathcal{L}$ is an event structure E together with a total function $\ell_E : |E| \to \mathcal{L}$. For $L \subseteq \mathcal{L}$, we define $E \backslash L$ as the restriction of E to the set of events $\{e \mid \forall e' \leq e, \ell_E(e') \notin L\}$. In other words, we remove every event with a label in L, and every event causally dependent on it. Maps of labelled event structures are required to preserve labels.*

We define the labels on $\mathit{Seq}(E)$ from the labels on E as $\ell_{\mathit{Seq}(E)}(e) := \ell_E(\pi_1(e))$. We define the labels of $A \times_\star B$ using the following synchronisation operation $\bullet : \mathbb{A} \cup \{\star\} \times \mathbb{A} \cup \{\star\} \to \mathbb{A} \cup \{\mathbf{0}\}$, where $\star$ stands for "undefined": $\ell_{A \times_\star B}(e) = \ell_A(\pi_1^\star(e)) \bullet \ell_B(\pi_2^\star(e))$

$$\begin{array}{l} a \bullet \bar{a} = \tau \\ \bar{a} \bullet a = \tau \end{array} \text{ for } a \in \mathit{Chan} \qquad \begin{array}{l} \alpha \bullet \star = \alpha \\ \star \bullet \alpha = \alpha \end{array} \text{ for } \alpha \in \mathbb{A} \qquad \alpha \bullet \beta = \mathbf{0} \text{ otherwise}$$

Figure 5b describes the concurrent semantics of the language. For any process P, we write $[\![P]\!]_{\mathrm{cs}}$ for the (potentially non-sequential, potentially infinite) event structure with labels in $\mathbb{A}$ representing P. See Figure 6 for an example. On top of the previous operations, we need some additional operations on labelled event structures:

- To represent nondeterministic choice, we need to put two event structures in parallel, while allowing only one of them to be used. For A and B two event structures, we define $A \# B$ as the event structure with $|A| \uplus |B|$ as events, with the same order, conflict and labels as in A and in B, but with every event of A being in conflict with every event of B.
- To represent prefixes, we need to create a new event. We write $\downarrow_{e:a} E$ for the event structure with events $|E| \uplus \{e\}$, with e labelled by a being the minimal event for $\leq_{\downarrow_{e:a} E}$, everything else being the same as in E.
- To represent process recursion, we use a least fixpoint. The order used for that least fixpoint is given by "substructure maps", i.e. total maps that are an inclusion on events, and preserve and reflect order and conflicts.

As claimed before, we can recover the interleaving semantics from the truly concurrent one. The proof of this theorem relies on the fact that *Seq* forms a coreflection.

▶ **Theorem 9** (Factorisation [17]). *For any process P of CCS, we have $[\![P]\!]_{is} \cong \mathit{Seq}([\![P]\!]_{cs})$.*

$$\frac{}{a.P \xrightarrow{a} P} \qquad \frac{P \xrightarrow{a} P'}{(P \varovee Q) \xrightarrow{a} P'}$$

$$\frac{a \notin \{c, \bar{c}\} \quad P \xrightarrow{a} P'}{\nu c.P \xrightarrow{a} \nu c.P'} \qquad \frac{P \xrightarrow{a} P'}{(Q \varovee P) \xrightarrow{a} P'}$$

$$\frac{P \xrightarrow{c} P' \quad Q \xrightarrow{\bar{c}} Q'}{(P|Q) \xrightarrow{\tau} (P'|Q')} \qquad \frac{P \xrightarrow{a} P'}{(P|Q) \xrightarrow{a} (P'|Q)}$$

$$\frac{P\{X \leftarrow \mu X.P\} \xrightarrow{a} Q}{\mu X.P \xrightarrow{a} Q} \qquad \frac{P \xrightarrow{a} P'}{(Q|P) \xrightarrow{a} (Q|P')}$$

(a) Interleaving semantics, using LTS.

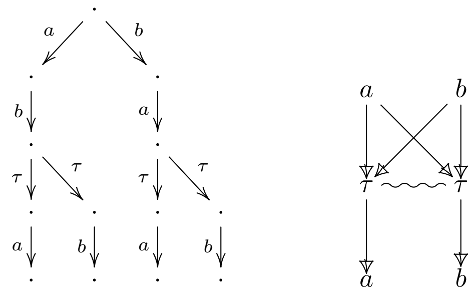
$$\begin{aligned}
[\![\mathbf{0}]\!]_{\mathrm{cs}} &= \varnothing \\
[\![(P_1|P_2)]\!]_{\mathrm{cs}} &= [\![P_1]\!]_{\mathrm{cs}} \times_\star [\![P_2]\!]_{\mathrm{cs}} \backslash \{\mathbf{0}\} \\
[\![(P_1 \varovee P_2)]\!]_{\mathrm{cs}} &= [\![P_1]\!]_{\mathrm{cs}} \mathbin{\#} [\![P_2]\!]_{\mathrm{cs}} \\
[\![a.P]\!]_{\mathrm{cs}} &= \downarrow_{P:a} [\![P]\!]_{\mathrm{cs}} \\
[\![\nu c.P]\!]_{\mathrm{cs}} &= [\![P]\!]_{\mathrm{cs}} \backslash \{c, \bar{c}\} \\
[\![\mu X.P]\!]_{\mathrm{cs}} &= [\![P\{X \leftarrow \mu X.P\}]\!]_{\mathrm{cs}}
\end{aligned}$$

(b) Concurrent semantics, using event structures.

Figure 5 Semantics of CCS, with $a \in \mathbb{A}$ and $c \in \mathit{Chan}$.

$\nu c. \left(a. \left(c.a \varovee c.b\right) \ \middle|\ b.\bar{c}\right)$

(a) A CCS process.



(b) Its interleaving semantics: a labelled tree.

(c) Its concurrent semantics: a labelled event structure.

Figure 6

2.4 Probabilistic CCS

The goal of this paper is to extend existing results on CCS to Probabilistic CCS, which is CCS enriched with a probabilistic choice[3] $\sum_{i \in I} p_i \cdot a_i.P_i$, with I a possibly infinite set, $\forall i \in I, 0 < p_i \leq 1$, $a_i \in \mathbb{A}$, and $\sum_{i \in I} p_i \leq 1$. We also assume the pairs (a_i, P_i) to be pairwise distinct. Notice that we allow sub-probabilistic sums like $\frac{1}{2} \cdot a.P$, but we forbid unguarded sums like $\frac{1}{2} \cdot (P_1|P_2) + \frac{1}{2} \cdot Q$. This guard restriction is similar to that found in the probabilistic π-calculus [8, 16]. The interleaving semantics we present here uses Segala automata [13, 14], and relies on this absence of unguarded sums to be well-defined. In this semantics, reductions are interpreted by an alternation of probabilistic choices and nondeterministic choices. Mathematically, the reduction $\to$ is a subset of $\mathbb{P} \times \mathcal{D}(\mathbb{A} \times \mathbb{P})$, where $\mathcal{D}(U)$ is the set of discrete sub-probability distributions over U. So for every process P, there is first a nondeterministic choice of $P \to S$, and then a probabilistic choice inside $S = \{(p_i, a_i, P_i) \mid i \in I\}$ of the action a_i and the reduced process P_i. Figure 7 describes this interleaving semantics.

3 Mixed Event Structures

We now develop the main contribution of this paper: an event structure model able to represent mixed internal and external choices, that we will use in the last section to give a concurrent semantics to the mixed probabilistic and nondeterministic choices of PCCS.

[3] Some papers [2] use a less general sum where the a_i are assumed to be equal.

$$\frac{a \in \mathbb{A}}{a.P \to \{(1,a,P)\}} \qquad \frac{}{\sum_{i\in I} p_i \cdot a_i.P_i \to \{(p_i,a_i,P_i) \mid i \in I\}} \qquad \frac{P \to S}{P \ovee Q \to S} \qquad \frac{Q \to S}{P \ovee Q \to S}$$

$$\frac{P\{X \leftarrow \mu X.P\} \to S}{\mu X.P \to S} \qquad \frac{P \to \{(p_i,a_i,P_i) \mid i \in I\}}{\nu c.P \to \{(p_i,a_i,P_i) \mid i \in I, a_i \notin \{c,\bar{c}\}\}} \qquad \frac{P \to \{(p_i,a_i,P_i) \mid i \in I\}}{P|Q \to \{(p_i,a_i,P_i|Q) \mid i \in I\}}$$

$$\frac{P \to \{(p_i,a_i,P_i) \mid i \in I\} \qquad Q \to \{(q_j,b_j,Q_j) \mid j \in J\}}{P|Q \to \{(p_iq_j,\tau,P_i|Q_j) \mid i \in I, j \in J, \bar{a}_i = b_j\}} \qquad \frac{Q \to \{(q_j,b_j,Q_j) \mid j \in J\}}{P|Q \to \{(q_j,b_j,P|Q_j) \mid j \in J\}}$$

Figure 7 Interleaving semantics of PCCS, using Segala automata.

3.1 Definition

When trying to represent processes of PCCS in event structures, one could think of simply using the existing probabilistic event structures [20], which are event structures E together with a valuation $v_E : \mathcal{C}(E) \to (0,1]$ satisfying some well-chosen properties (Definition 21). However, looking at the two processes $(\frac{1}{2}a + \frac{1}{2}b)$ and $\left((\frac{1}{2}a) \ovee (\frac{1}{2}b)\right)$, we remark that:

- They have only two different computational events: "receiving on a" and "receiving on b"; which means that natural representations using event structures will only have two events.
- Those two computational events cannot occur together, so natural representations using event structures have those two events in conflict.
- Those two computational events are associated with probability half, which means that the valuation is necessarily $v(\{a\}) = \frac{1}{2} = v(\{b\})$

In other words, if we try to represent them using probabilistic event structures, both of them will be represented by the same structure. But they have very different interleaving semantics in Segala automata, so it would not be a sound representation. A similar example using full probabilistic distributions is $\left(\frac{1}{2}a + \frac{1}{2}b\right) \ovee \left(\frac{1}{2}c + \frac{1}{2}d\right)\big)$ and $\left(\frac{1}{2}a + \frac{1}{2}c\right) \ovee \left(\frac{1}{2}b + \frac{1}{2}d\right)\big)$.

In order to distinguish these two processes, the solution we propose is to have two kinds of conflicts: an *external conflict*, used for the nondeterministic choice $\ovee$, and an *internal conflict*, used for the probabilistic choice $+$. In this section, we explore in more detail event structures with two kinds of conflicts, named *mixed event structures.*

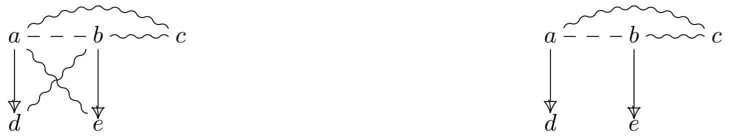
We will keep using the notation $\#$ for the union of both conflicts, and we will use the notation $\square$ for internal conflicts and $\sharp$ for external conflicts. Since we have two kinds of conflicts, we will have two kinds of "configurations": the usual set of configurations, computed from $\#$, inside which no conflicts are tolerated, and the set of *worlds* inside which internal conflicts $\square$ are accepted. These considerations give rise to the following definition.

▶ **Definition 10.** *A* mixed event structure *(mes) E is a quadruple $(|E|, \leq_E, \#_E, \sharp_E)$ where*

- *$\mathcal{U}(E) = (|E|, \leq_E, \#_E)$ is an event structure. We write $\mathcal{C}(E) = \mathcal{C}(\mathcal{U}(E))$ for the set of* configurations. *We define $\leftrightsquigarrow_E, \rightarrowtriangle_E, [e], [e)$ as previously.*
- *$(|E|, \leq_E, \sharp_E)$ is an event structure. We write $\mathcal{W}(E) = \mathcal{C}(|E|, \leq_E, \sharp_E)$ for the set of* worlds.
- *$\sharp_E \subseteq \#_E$, or equivalently $\mathcal{C}(E) \subseteq \mathcal{W}(E)$.*

We define the internal conflict $\square_E$ as $(\#_E \backslash \sharp_E)$.

The internal conflict $\square_E$ is a symmetric irreflexive relation, but $(|E|, \leq_E, \square_E)$ may not be an event structure. In fact, we will later (Definition 16) consider the special case $\square_E \subseteq \leftrightsquigarrow_E$. The operation $\mathcal{U}$ turns out to be the right adjoint in a coreflection (Theorem 13) between mes and event structures. Graphically, we use dashes to represent minimal internal conflict (i.e. $\leftrightsquigarrow_E \cap \square_E$), wiggly lines to represent the minimal conflict of $(|E|, \leq_E, \sharp_E)$, and arrows to represent $\rightarrowtriangle_E$. Those are enough to characterise all the conflicts. For example, in Figure 8, we can deduce $c \sharp e$ in both E_1 and E_2, in E_1 we also have $d \sharp e$ while in E_2 we have $d \square e$.



$\mathcal{C}(E_1) = \{\varnothing, \{a\}, \{b\}, \{c\}, \{a,d\}, \{b,e\}\}$
$\mathcal{W}(E_1) = \mathcal{C}(E_1) \cup \{\{a,b\}\}$

$\mathcal{C}(E_2) = \mathcal{C}(E_1)$
$\mathcal{W}(E_2) = \mathcal{W}(E_1) \cup \{\{a,b,d\}, \{a,b,e\}, \{a,b,d,e\}\}$

Figure 8 Mixed event structures E_1 and E_2.

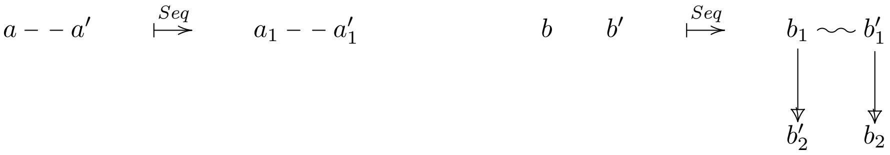


Figure 9 Image by the sequentialisation of the mes A and B.

Our goal is to extend all the operations previously defined on event structures. So we want mes to have (a)synchronous products, and a sequentialisation *Seq* functor forming a coreflection. In particular, it means sequentialising a mes which is already sequential (Definition 16) should be an isomorphism, so in Figure 9, $\mathit{Seq}(A) \cong A$. Moreover, we want this extension to be conservative: a mes with $\square = \varnothing$ (i.e. $\mathcal{W} = \mathcal{C}$) should behave as an event structure. In particular, the sequentialisation of an event structure should be preserved by the inclusion of event structures into mes. So in Figure 9, $\mathit{Seq}(B)$, where B is seen as a mes, should be the same as $\mathit{Seq}(B)$ where B is seen as an event structure.

We also want (a)synchronous products of mes to correspond to (a)synchronous products of event structures. As previously, we use a coreflection to express this preservation. The right adjoint to the inclusion will be the forgetful functor $\mathcal{U}$. But to talk about the coreflection, we first need to define the category of mes. We will provide justification for each of the 4 conditions on maps of mes (Definition 11).

Firstly, since we want $\mathcal{U}$ to be a functor, then for every map of mes f from A to B, there corresponds a map of event structures $\mathcal{U}(f)$ from $\mathcal{U}(A)$ to $\mathcal{U}(B)$.

Secondly, since we want $\mathcal{U}$ to form a coreflection, for every mes A, we need a map from $\mathcal{U}(A)$ to A which is the identity function on events. This implies that we have an identity-on-events map from the mes with two events $a \# b$ to the mes with two events $a \square b$. Since we do not want these two mes to be isomorphic, we do not want any identity-on-events map from $a \square b$ to $a \# b$. This leads to the second condition on maps of mes: maps of mes preserve worlds.

Thirdly, considering Figure 9 again, any total map g from $\mathit{Seq}(A)$ to $\mathit{Seq}(B)$ has to send the world $\{a_1, a'_1\}$ to a world, hence necessarily $g(a_1) = g(a'_1)$. It follows that for *Seq* to be a functor, any map f from A to B should respect $f(a) = f(a')$. So we need to forbid $f(a) = b$ and $f(a') = b'$ from forming a map. Which leads to the following restriction: if $a \square a'$, f defined on a and a', and $f(a) \neq f(a')$, then $f(a) \square f(a')$.

Lastly, due to some more subtle problems in the case of partial maps (see the full version), we need the domain of the map to be closed under $\square$.

▶ **Definition 11.** *A map f of mes from E to F is a (possibly partial) function from $|E|$ to $|F|$ satisfying:*

map of event structures *f is a map from $\mathcal{U}(E)$ to $\mathcal{U}(F)$;*
world preserving *for every $w \in \mathcal{W}(E)$, $f(w) \in \mathcal{W}(F)$;*
$\square$-preserving *for every $a \square_E b$ where f is defined and $f(a) \neq f(b)$, then $f(a) \square_F f(b)$; and*
$\square$-equidefinability *for every $a \square_E b$, f is defined on both or none.*

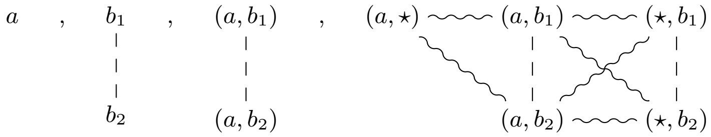


Figure 10 The mes A, B, $A \times B$ and $A \times_\star B$.

▶ **Proposition 12.** *Mes and partial maps of mes form a category, written* $\mathsf{MES}_\star$*, with the empty event structure as a terminal object. Mes and total maps of mes form a subcategory, written* MES*.*

▶ **Theorem 13.** *The functor* $\mathcal{U}$ *is a left adjoint to the inclusion, forming a coreflection.*

$$\begin{tikzcd} \mathsf{ES} \arrow[r, bend left, "\hookrightarrow"] \arrow[r, phantom, "\perp"] & \mathsf{MES} \arrow[l, bend left, "\mathcal{U}"] \end{tikzcd} \qquad \begin{tikzcd} \mathsf{ES}_\star \arrow[r, bend left, "\hookrightarrow"] \arrow[r, phantom, "\perp"] & \mathsf{MES}_\star \arrow[l, bend left, "\mathcal{U}"] \end{tikzcd}$$

Additionally, the inclusion preserves categorical limits.

See the full version for the proof. This theorem implies that both the forgetful functor $\mathcal{U}$ and the inclusion functor ↪ preserve (a)synchronous products.

3.2 Products and Sequentiality

From the previous coreflection, we know that mes coming from event structures have synchronous and asynchronous products. These constructions extend to all mes. The proof is technical, and can be found in the full version.

▶ **Proposition 14.** $\mathsf{MES}_\star$ *is a cartesian category, with* $\times_\star$ *as a product and* $\varnothing$ *as a unit.* MES *has a product* $\times$*.*

As a remark, since we have $\mathcal{U}(A \times_\star B) = \mathcal{U}(A) \times_\star \mathcal{U}(B)$, we know that the events, order, and conflict of the mes $A \times_\star B$ are the same as those of the event structure $\mathcal{U}(A) \times_\star \mathcal{U}(B)$. And we have a similar property for $A \times B$. One can then characterise the internal conflict:

▶ **Proposition 15.** *For* E_1 *and* E_2 *two mes, with* $P = E_1 \times E_2$ *and* $P_\star = E_1 \times_\star E_2$*, we have:*

$$a \,\square_P\, b \iff [a] \cup [b), [a) \cup [b] \in \mathcal{W}(P) \text{ and } \forall i \in \{1,2\}, \begin{cases} \pi_i(a) = \pi_i(b), \text{ or} \\ \pi_i(a) \,\square_{E_i}\, \pi_i(b) \end{cases}$$

$$a \,\square_{P_\star}\, b \iff [a] \cup [b), [a) \cup [b] \in \mathcal{W}(P_\star) \text{ and } \forall i \in \{1,2\}, \begin{cases} \pi_i^\star \text{ undef. on } \{a,b\}, \text{ or} \\ \pi_i^\star(a) = \pi_i^\star(b), \text{ or} \\ \pi_i^\star(a) \,\square_{E_i}\, \pi_i^\star(b) \end{cases}$$

This characterisation is recursive, since it refers to worlds and worlds are defined by the internal conflict. However, this is a well-founded recursion[4], so this proposition could be used as a definition of the (a)synchronous product. Figure 10 is an example of the synchronous and asynchronous product of two mes.

We now aim to extend the definition of sequentiality to mes. However, a problem arises: the usual definition of sequentiality "$\forall a, b \in c \in \mathcal{C}(E) \implies a \leq b$ or $b \leq a$" gives only a many-to-one correspondence with trees, and is no longer equivalent to the categorical

[4] The well-founded relation is $(a',b') \prec (a,b) \iff a' \leq_{E_1 \times_\star E_2} a$ and $b' \leq_{E_1 \times_\star E_2} b$ and $(a',b') \neq (a,b)$.

definition "there exists a total map from E to $\mathcal{N}$". Indeed, since maps are expected to be $\square$-preserving, the mes E_2 from Figure 8 satisfies the first but not the second. So we strengthen the first definition so that it is equivalent to the second: we require internal conflicts to be minimal, i.e. $\square \subseteq \sim\!\sim$. This allow to recover a one-to-one correspondence with alternating trees (Proposition 20).

▶ **Definition 16.** *A mes E is* sequential *if it satisfies one of the following equivalent properties:*

- *E is forest-shaped, with branches being in conflict with each other, and $\square_E \subseteq \sim\!\sim_E$.*
- *For every $a, b \in x \in \mathcal{C}(E)$, either $a \leq_E b$ or $b \leq_E a$. Moreover, $\square_E \subseteq \sim\!\sim_E$.*
- *There exists a (necessarily unique) total map from E to $\mathcal{N}$.*
- *E is isomorphic to $E \times \mathcal{N}$.*

Analogously to the correspondence between trees and event structures, *alternating trees* (Definition 18) will correspond to sequential mes.

▶ **Proposition 17.** *We write* Seq-MES *(resp.* Seq-MES$_\star$*) for the subcategory of* MES *(resp.* MES$_\star$*) with only sequential mes. It is a cartesian category, with $\times$ (resp. $\times_\star$) as a product and $\mathcal{N}$ (resp. $\varnothing$) as a terminal object.*

Now, we define the sequentialisation as previously: for a mes E, we define the sequential mes $\mathit{Seq}(E)$ as $E \times \mathcal{N}$. The coreflection (Theorem 6) still holds in the case of mes. A proof can be found in the full version.

3.3 Labelled Mixed Event Structures

As in the case of event structures, one can add labels to mes. A *mes labelled in $\mathcal{L}$* is a mes E together with a labelling function $\ell_E : |E| \to \mathcal{L}$. Maps are required to preserve labels, *Seq* preserves labels, and we compute the labels of asynchronous product using $\bullet$.

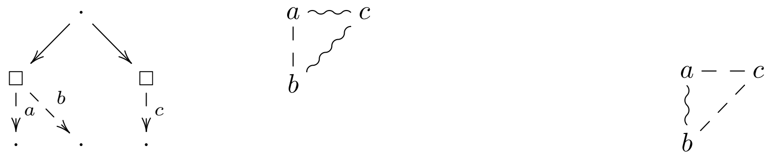
3.4 Alternating Trees

Just as we can choose a root node of an LTS and unfold it into a tree, so can we choose a root node in a Segala automaton and unfold it into a Segala tree. We define here *labelled alternating trees* which can be understood as "Segala trees, but without probabilities"; the complete definition of a Segala tree will come later (Definition 24), and we will omit the definition of Segala automaton [13, 14].

▶ **Definition 18.** *A* labelled alternating tree *is a tree such that:*

- *Nodes of even depth (including the root) are called* states*. All leaves must be states.*
- *Nodes of odd depth are called* intermediary positions.
- *Transitions from intermediary positions to states are labelled. They are called* internal transitions *and written $\dashrightarrow_\ell$ if labelled by ℓ.*
- *Transitions from states to intermediary positions are unlabelled. They are called* external transitions *and written $\to$.*

Labelled alternating trees can be represented directly by labelled sequential mes. Figure 11a shows an alternating tree and its representation by a labelled sequential mes. Every intermediary position of the labelled alternating tree corresponds to a *cell* (set of events that are pairwise related by $\square$), and each labelled transition from this intermediary position corresponds to a labelled event of this cell. However, this is not a one-to-one correspondence, since labelled alternating trees will only generate mes where $\square$ is a transitive relation. In particular, the labelled sequential mes in Figure 11b cannot be built using labelled alternating trees.



(a) A alternating tree and its sequential mes. **(b)** A sequential mes corresponding to no alternating tree.

Figure 11 Alternating trees and mes.

▶ **Definition 19.** *A mes E is said to be* locally transitive *if for every $x \overset{a}{\relbar\!\!\subset}$, $x \overset{b}{\relbar\!\!\subset}$, and $x \overset{c}{\relbar\!\!\subset}$ with $x \in \mathcal{C}(E)$ and a, b, c pairwise distinct, we have $a \mathrel{\square_E} b \mathrel{\square_E} c \Rightarrow a \mathrel{\square_E} c$.*

We use this more restricted notion of transitivity because "$\square$ is transitive" is not a property stable under (a)synchronous products, while local transitivity is: the (a)synchronous product of two locally transitive mes is locally transitive. In fact every definition and result previously stated still applies if we restrict ourselves to locally transitive mes.

▶ **Proposition 20.** *There is a one-to-one correspondence between labelled alternating trees and labelled locally transitive sequential mes. Moreover, for a mes E, Seq(E) corresponds to an alternating tree if and only if E is locally transitive.*

4 Concurrent Semantics of Probabilistic CCS

The goal of this section is to give a concurrent semantics to Probabilistic CCS, with a factorisation property similar to the one of CCS. We first add a probabilistic valuation to mes, in order to relate them to Segala trees, used for the interleaving semantics of PCCS. Then, we describe how to extend the concurrent semantics for CCS into one for PCCS.

4.1 Mixed Probabilisitic Event Structures

We recall some notions of probabilistic event structures [15, 20]. They are event structures together with a probability valuation $v : \mathcal{C}(E) \to (0, 1]$, interpreted as *the probability of reaching at least this configuration*, such that $v(\varnothing) = 1$ and a condition of *monotonicity* (Definition 21). Simple consequences of the condition of *monotonicity* are:

- The valuation v is decreasing: $x \subseteq y \implies v(x) \geq v(y)$
- Events in conflict have conditional probability whose sum is less than or equal to one: $\forall 1 \leq i \leq n, x \overset{a_i}{\relbar\!\!\subset}$ and $\forall 1 \leq i < j \leq n, a_i \mathrel{\#} a_j \implies \sum_{i=1}^{n} \frac{v(x \cup \{a_i\})}{v(x)} \leq 1$
- Events not in conflict respect a variant of the inclusion-exclusion principle: $x, y, x \cap y, x \cup y \in \mathcal{C}(E) \implies v(x \cap y) - v(x) - v(y) + v(x \cup y) \geq 0$

We want to add similar conditions to mes. We consider a mes with two events a and b. We aim to use the internal conflict $\square$ of mes for probabilistic choices $+$ of PCCS, meaning that if $a \mathrel{\square} b$, we can expect $v(\{a\}) + v(\{b\}) \leq 1$, so they respect the *monotone* condition. However, since we aim to use the external conflict ⋕ for the nondeterministic choice $\circledvee$ of PCCS, if a ⋕ b, we can have $v(\{a\}) = 1 = v(\{b\})$, which does not respect the *monotone* condition. This guides us to the following condition: within worlds, conflicts are necessarily $\square$, so the *monotone* condition has to be respected, however no condition is expected to hold across different worlds.

▶ **Definition 21.** *A* mixed probabilisitic event structure *(mpes) is a mes E together with a probabilistic valuation $v_E : \mathcal{C}(E) \to (0, 1]$ such that:*

Initialised $v_E(\varnothing) = 1$
Monotone *For $x, y_1, \ldots, y_n \in \mathcal{C}(E)$, with $\forall 1 \leq i \leq n, x \subseteq y_i$, we write:*

- *For $\varnothing \neq I \subseteq \{1, \ldots, n\}, y_I := \bigcup_{i \in I} y_i$*
- *For $X \notin \mathcal{C}(E), v_E(X) := 0$*
- *$d(x; y_1, \ldots, y_n) := v_E(x) - \sum_{\varnothing \neq I \subseteq \{1,\ldots,n\}} (-1)^{|I|+1} v_E(y_I)$*

We then demand that $y_{\{1,\ldots,n\}} \in \mathcal{W}(E) \implies d(x; y_1, \ldots, y_n) \geq 0$

It is labelled *if the underlying mes is labelled. It is* sequential *if the underlying mes is sequential. A map of mpes is a map of mes $f : A \rightharpoonup B$ such that $\forall x \in \mathcal{C}(A), v_A(x) \leq v_B(f(x))$.*

Unlike [20], we use a valuation that is strictly positive on configurations instead of positive or null. This choice come from the absence of 0-probability branches in Segala trees.

We write $\mathsf{MPES}_\star$, MPES, $\mathsf{Seq\text{-}MPES}_\star$, and $\mathsf{Seq\text{-}MPES}$ for the categories of mpes with maps restricted to total ones and/or with objects restricted to sequential ones. The empty mpes $\varnothing$ is a terminal object for $\mathsf{MPES}_\star$ and $\mathsf{Seq\text{-}MPES}_\star$, while $\mathcal{N} = (\mathbb{N}, \leq, \varnothing, \varnothing, v_\mathcal{N} : x \mapsto 1)$ is a terminal object for $\mathsf{Seq\text{-}MPES}$.

We now want to extend the (a)synchronous product to mpes, but it is unfortunately not always possible to preserve the universal property of (a)synchronous products.

▶ **Proposition 22.** MPES *and* $\mathsf{MPES}_\star$ *do not have all products.*

A counterexample can be found in the full version. While no categorical product can be defined, both MPES and $\mathsf{MPES}_\star$ have symmetric monoidal products $\otimes$ and $\otimes_\star$ which generalise $\times$ and $\times_\star$ of mes. The underlying problem of the absence of products is a problem of *probabilistic correlation*. We fail to define $A \times B$ because the correlations between the events of A and the events of B is unspecified. So when defining $A \otimes B$, we make the most canonical choice and consider that A and B are probabilistically independent. This choice is also consistent with the processes of PCCS: when considering $(P|Q)$, the probabilistic choices inside P are assumed to be independent of the probabilistic choices inside Q.

▶ **Definition 23.** *For A and B two mpes, we define $A \otimes_\star B$ as follows:*

- *The underlying mes is $A \times_\star B$, where A and B are seen as mes.*
- *$\forall x \in \mathcal{C}(A \otimes_\star B), v_{A \otimes_\star B}(x) = v_A(\pi_1^\star(x)) \cdot v_B(\pi_2^\star(x))$.*

This is a mpes (proof in the full version). We define $A \otimes B$ similarly.

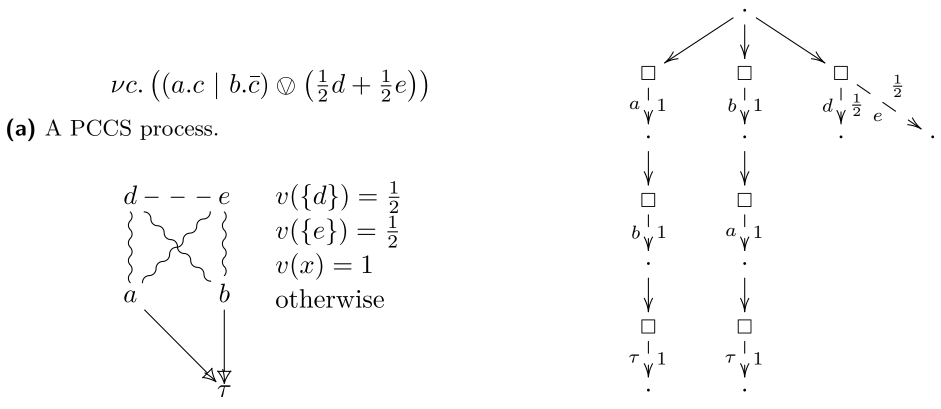
A remarkable property is that when A (or B) respects $\forall x \in \mathcal{C}(A), v_A(x) = 1$, then $A \otimes_\star B$ is a product in $\mathsf{MPES}_\star$, and $A \otimes B$ too in MPES. The sequentialisation $\mathit{Seq}(E) := E \otimes \mathcal{N}$ still induces a coreflection.

We stated earlier that alternating trees were "Segala trees without probabilities", and we will now define Segala trees, and explain their correspondence with locally transitive sequential mpes.

▶ **Definition 24.** *A Segala tree labelled by $\mathcal{L}$ is an* alternating tree *labelled by $(0, 1] \times \mathcal{L}$, such that if we have $i \dashrightarrow_{(p_k, a_k)} s_k$ for $1 \leq k \leq n$ (with s_k being distinct), then $\sum_{k=1}^{n} p_k \leq 1$.*

▶ **Theorem 25.** *There is a one-to-one correspondence between Segala trees and labelled locally transitive sequential mpes.*

This correspondence is given by the correspondence between labelled alternating trees and labelled locally transitive sequential mes, with the probability of a transition $\dashrightarrow_{(p,a)}$ corresponding to an event e given by $p = \frac{v([e])}{v([e))}$, and reciprocally the valuation of a configuration x being the product of all the probabilities on transitions of the Segala tree along the branch corresponding to x.



$$\nu c.\,\big((a.c \mid b.\bar{c}) \otimes \big(\tfrac{1}{2}d + \tfrac{1}{2}e\big)\big)$$

(a) A PCCS process.

(b) Its concurrent semantics:
a labelled mixed probabilisitic event structure.

(c) Its interleaving semantics:
a Segala tree.

Figure 12

4.2 Concurrent Semantics of PCCS

In order to define the concurrent semantics of PCCS, we first extend the constructors on event structures used for CCS to mpes.

- $\mathcal{U}(E\backslash L) = \mathcal{U}(E)\backslash L$; $\boxdot_{E\backslash L}$ and $v_{E\backslash L}$ are the restriction of $\boxdot_E$ and v_E to $|E\backslash L|$.
- $\mathcal{U}(A \mathbin{\#} B) = \mathcal{U}(A) \mathbin{\#} \mathcal{U}(B)$; $\boxdot_{A \# B}$ is the disjoint union, and $v_{A \# B}$ is the co-pairing.
- $\mathcal{U}(\downarrow_{e:a} E) = \downarrow_{e:a} \mathcal{U}(E)$; $\boxdot_{\downarrow_{e:a} E}$ and $v_{\downarrow_{e:a} E}$ are the extension of $\boxdot_E$ and v_E to $|\downarrow_{e:a} E|$ (so in particular $v_{\downarrow_{e:a} E}(\{e\}) = v_E(\varnothing) = 1$)
- Recursion is still a least fixpoint for the "substructure maps", i.e. total maps that are an inclusion on events, and preserve and reflect order, internal and external conflicts, and the valuation.

This allows us to represent every process of CCS as an mpes, using the same notation as in Figure 5b (with $\otimes$ instead of $\times$). We define the operation $+$ on mpes as follows. For $(p_i)_{i\in I} \in (0,1]^I$ and $\sum_{i\in I} p_i \leq 1$, we define $S = \sum_{i\in I} p_i \cdot E_i$ with:

- $\mathcal{U}(S) = \#_{i\in I}\mathcal{U}(E_i)$ and $v_S(x) = p_i \cdot v_{E_i}(x)$ if $x \in \mathcal{C}(E_i)$
- $e \boxdot_S e' \iff \exists i \in I, \begin{cases} e \boxdot_{E_i} e', \text{ and} \\ e, e' \in |E_i| \end{cases}$ or $\exists i \neq j \in I, \begin{cases} e \in |E_i|, \text{ and} \\ e' \in |E_j| \end{cases}$

The concurrent semantics $[\![\_]\!]_{\mathrm{cs}}$ of PCCS is given by Figure 5b together with this additional rule: $[\![\sum_{i\in I} p_i \cdot a_i.P_i]\!]_{\mathrm{cs}} := \sum_{i\in I} p_i \cdot [\![a_i.P_i]\!]_{\mathrm{cs}}$. See Figure 12 for an example.

We note that for every process P of PCCS, $[\![P]\!]_{\mathrm{cs}}$ is locally transitive, meaning that its sequentialisation will correspond to a Segala tree. If we write $[\![P]\!]_{\mathrm{is}}$ for the semantics of the process P with Segala trees, seen as labelled, locally transitive, sequential mpes, then we have the following factorisation theorem.

▶ **Theorem 26** (Factorisation). *For any process P of PCCS, we have $[\![P]\!]_{is} \cong Seq([\![P]\!]_{cs})$.*

The proof of this theorem relies on the sequentialisation coreflection, which ensures that the interpretation of the parallel composition is preserved.

This concludes the main contribution of this paper: we have built a concurrent model able to represent both probabilistic and nondeterministic choices, and used it to give to PCCS a concurrent semantics compatible with its usual interleaving semantics.

As future work, we wish to investigate PCCS without the guard restriction, hence allowing *unguarded probabilistic choice* $\sum_{i \in I} p_i \cdot P_i$. We are able to give to Unguarded PCCS a concurrent semantics using potentially non locally transitive mpes, however the relationship with existing interleaving semantics remains unclear.

References

1 Jos C. M. Baeten. A brief history of process algebra. *Theor. Comput. Sci.*, 335(2-3):131–146, 2005. doi:10.1016/j.tcs.2004.07.036.

2 Christel Baier and Marta Z. Kwiatkowska. Domain equations for probabilistic processes. *Electr. Notes Theor. Comput. Sci.*, 7:34–54, 1997. doi:10.1016/S1571-0661(05)80465-7.

3 Jan A. Bergstra and Jan Willem Klop. Process Algebra for Synchronous Communication. *Information and Control*, 60(1-3):109–137, 1984. doi:10.1016/S0019-9958(84)80025-X.

4 Simon Castellan. Weak memory models using event structures. In Julien Signoles, editor, *Vingt-septièmes Journées Francophones des Langages Applicatifs (JFLA 2016)*, Saint-Malo, France, January 2016. URL: https://hal.inria.fr/hal-01333582.

5 Simon Castellan, Pierre Clairambault, Hugo Paquet, and Glynn Winskel. The Concurrent Game Semantics of Probabilistic PCF. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, pages 215–224, New York, NY, USA, 2018. ACM. doi:10.1145/3209108.3209187.

6 Pierre Clairambault, Marc de Visme, and Glynn Winskel. Game semantics for quantum programming. *PACMPL*, 3(POPL):32:1–32:29, 2019. URL: https://dl.acm.org/citation.cfm?id=3290345, doi:10.1145/3290345.

7 Jean Goubault-Larrecq. Isomorphism theorems between models of mixed choice. *Mathematical Structures in Computer Science*, 27(6):1032–1067, 2017. doi:10.1017/S0960129515000547.

8 Oltea Mihaela Herescu and Catuscia Palamidessi. Probabilistic asynchronous pi-calculus. *CoRR*, cs.PL/0109002, 2001. arXiv:cs.PL/0109002.

9 Gilles Kahn, editor. *Semantics of Concurrent Computation, Proceedings of the International Symposium, Evian, France, July 2-4, 1979*, volume 70 of *Lecture Notes in Computer Science*. Springer, 1979. doi:10.1007/BFb0022459.

10 Robin Milner. Lectures on a Calculus for Communicating Systems. In *Seminar on Concurrency, Carnegie-Mellon University, Pittsburg, PA, USA, July 9-11, 1984*, pages 197–220, 1984. doi:10.1007/3-540-15670-4_10.

11 Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri Nets, Event Structures and Domains. In Gilles Kahn, editor, *Semantics of Concurrent Computation, Proceedings of the International Symposium, Evian, France, July 2-4, 1979*, volume 70 of *Lecture Notes in Computer Science*, pages 266–284. Springer, 1979. doi:10.1007/BFb0022474.

12 Mogens Nielsen and Erik Meineche Schmidt, editors. *Automata, Languages and Programming, 9th Colloquium, Aarhus, Denmark, July 12-16, 1982, Proceedings*, volume 140 of *Lecture Notes in Computer Science*. Springer, 1982. doi:10.1007/BFb0012751.

13 Roberto Segala. *Modeling and verification of randomized distributed real-time systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1995. URL: http://hdl.handle.net/1721.1/36560.

14 Ana Sokolova and Erik P. de Vink. Probabilistic Automata: System Types, Parallel Composition and Comparison. In *Validation of Stochastic Systems - A Guide to Current Research*, pages 1–43, 2004. doi:10.1007/978-3-540-24611-4_1.

15 Daniele Varacca, Hagen Völzer, and Glynn Winskel. Probabilistic event structures and domains. *Theor. Comput. Sci.*, 358(2-3):173–199, 2006. doi:10.1016/j.tcs.2006.01.015.

16 Daniele Varacca and Nobuko Yoshida. Probabilistic pi-Calculus and Event Structures. *Electr. Notes Theor. Comput. Sci.*, 190(3):147–166, 2007. doi:10.1016/j.entcs.2007.07.009.

17 Glynn Winskel. Event Structure Semantics for CCS and Related Languages. In Mogens Nielsen and Erik Meineche Schmidt, editors, *Automata, Languages and Programming, 9th Colloquium, Aarhus, Denmark, July 12-16, 1982, Proceedings*, volume 140 of *Lecture Notes in Computer Science*, pages 561–576. Springer, 1982. doi:10.1007/BFb0012800.

18 Glynn Winskel. Event Structures. In *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part II, Proceedings of an Advanced Course, Bad Honnef, Germany, 8-19 September 1986*, pages 325–392, 1986. doi:10.1007/3-540-17906-2_31.

19 Glynn Winskel. Event Structures with Symmetry. *Electr. Notes Theor. Comput. Sci.*, 172:611–652, 2007. doi:10.1016/j.entcs.2007.02.022.

20 Glynn Winskel. Probabilistic and Quantum Event Structures. In *Horizons of the Mind. A Tribute to Prakash Panangaden - Essays Dedicated to Prakash Panangaden on the Occasion of His 60th Birthday*, pages 476–497, 2014. doi:10.1007/978-3-319-06880-0_25.

Verification of Flat FIFO Systems

Alain Finkel
LSV, ENS Paris-Saclay, CNRS, Université Paris-Saclay, France
UMI ReLaX, French-Indian research laboratory in computer sciences, Chennaï, India

M. Praveen
Chennai Mathematical Institute, India
UMI ReLaX, French-Indian research laboratory in computer sciences, Chennaï, India

Abstract

The decidability and complexity of reachability problems and model-checking for flat counter systems have been explored in detail. However, only few results are known for flat FIFO systems, only in some particular cases (a single loop or a single bounded expression). We prove, by establishing reductions between properties, and by reducing SAT to a subset of these properties that many verification problems like reachability, non-termination, unboundedness are NP-complete for flat FIFO systems, generalizing similar existing results for flat counter systems. We construct a trace-flattable counter system that is bisimilar to a given flat FIFO system, which allows to model-check the original flat FIFO system. Our results lay the theoretical foundations and open the way to build a verification tool for (general) FIFO systems based on analysis of flat subsystems.

2012 ACM Subject Classification Theory of computation → Parallel computing models

Keywords and phrases Infinite state systems, FIFO, counters, flat systems, reachability, termination, complexity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.12

Funding The work reported was carried out in the framework of ReLaX, UMI2000 (ENS Paris-Saclay, CNRS, Univ. Bordeaux, CMI, IMSc). This work was also supported by the grant ANR-17-CE40-0028 of the French National Research Agency ANR (project BRAVAS).
M. Praveen: Partially supported by a grant from the Infosys foundation.

1 Introduction

FIFO systems. Asynchronous distributed processes communicating through First In First Out (FIFO) channels are used since the seventies as models for protocols [33], distributed and concurrent programming and more recently for web service choreography interface [12]. Since FIFO systems simulate counter machines, most reachability properties are *undecidable* for FIFO systems: for example, the basic task of checking if the number of messages buffered in a channel can grow unboundedly is undecidable [11].

There aren't many interesting and useful FIFO subclasses with a *decidable* reachability problem. Considering FIFO systems with a unique FIFO channel is not a useful restriction since they may simulate Turing machines [11]. A few examples of decidable subclasses are half-duplex systems [13] (but they are restricted to two machines since the natural extension to three machines leads to undecidability), existentially bounded deadlock free FIFO systems [26] (but it is undecidable to check if a system is existentially bounded, even for deadlock free FIFO systems), synchronisable FIFO systems (the property of synchronisability is undecidable [24] and moreover, it is not clear which properties of synchronisable systems are decidable), flat FIFO systems [6, 7] and lossy FIFO systems [1] (but one loses the perfect FIFO mechanism).

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 12; pp. 12:1–12:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Flat systems. A flat system [4, 23, 14, 5] is a system with a finite control structure such that every control-state belongs to at most one loop. Equivalently, the language of the control structure is included in a bounded language of the form $w_1^* w_2^* ... w_k^*$ where every w_i is a non empty word. Analyzing flat systems essentially reduces to accelerating loops (i.e., to compute finite representations of the effect of iterating each loop arbitrarily many times) and to connect these finite representations with one another. Flat systems are particularly interesting since one may under-approximate any system by its flat subsystems.

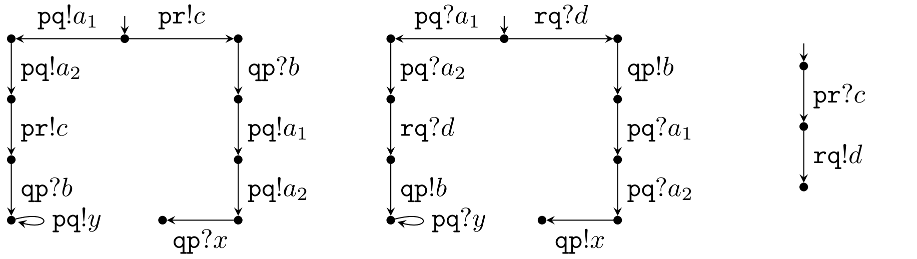
For counter systems [19, 28], this strategy lead to some tools like FAST [3], LASH, TREX [2], FLATA [10] which enumerate all flat subsystems till the reachability set is reached. This strategy is not an algorithm since it may never terminate on some inputs. However in practice, it terminates in many cases; e.g., in [3], 80% of the examples (including Petri nets and multi-threaded Java programs) could be effectively verified. The complexity of flat counter systems is well-known: reachability is NP-complete for variations of flat counter systems [27, 9, 18], model-checking first-order formulae and linear μ-calculus formulae is PSPACE-complete while model-checking Büchi automata is NP-complete [17]; equivalence between model-checking flat counter systems and Presburger arithmetic is established in [16].

Flat FIFO systems. We know almost nothing about flat FIFO systems, even the complexity of reachability is not known. Boigelot et al. [6] used recognizable languages (QDD) for representing FIFO channel contents and proved that the acceleration of *one-counting* loops (a loop is one-counting if it sends messages to only one channel), from an initial QDD, produces another computable QDD. Bouajjani and Habermehl [7] proved that the acceleration of *any* loop can be finitely represented by combining a deterministic flat finite automaton and a Presburger formula (CQDD) that are both computable. However, surprisingly, no upper bound for the Boigelot et al.'s and for the Bouajjani et al.'s loop-acceleration algorithms are known. Just the complexity of the inclusion problem for QDD, CQDD and SLRE (SLRE are both QDD and CQDD) are partially known (respectively PSPACE-complete, N2EXPTIME-hard, CONP-complete) [25]. But the complexity of the reachability problem for flat FIFO systems was not known. Only the complexity of the control-state reachability problem was known to be NP-complete for flat FIFO systems [21]. Moreover, other properties and model-checking have not been studied for flat FIFO systems.

Contributions. We solve the open problem of the complexity of the reachability problem for flat FIFO systems by showing that it is NP-complete; we extend this result to other usual verification properties and show that they are also NP-complete. Then we show that a flat FIFO system can be simulated by a synchronized product of counter systems. This synchronized product is flattable and its reachability set is semilinear.

2 Preliminaries

We write $\mathbb{Z}$ (resp. $\mathbb{N}$) to denote the set of integers (resp. non-negative integers). A finite alphabet is any finite set Σ. Its elements are referred to as letters; Σ^* is the set of all finite sequences of letters, referred to as words. We denote by $w_1 w_2$ the word obtained by concatenating w_1 and w_2; and ϵ is the empty sequence, which is the unity for the concatenation operation. We write Σ^+ for $\Sigma^* \setminus \{\epsilon\}$. If w_1 is a prefix of w_2, we denote by $w_1^{-1} w_2$ the word obtained from w_2 by dropping the prefix w_1. If w_1 is not a prefix of w_2, then $w_1^{-1} w_2$ is undefined. A word $z \in \Sigma^*$ is primitive if $z \notin w^* \setminus \{w\}$ for any $w \in \Sigma^*$. We



(a) Process P. **(b)** Process Q. **(c)** Process R.

Figure 1 FIFO system of Example 2.2.

denote by $Parikh(w) : \Sigma \to \mathbb{N}$ the function that maps each letter $a \in \Sigma$ to the number of times a occurs in w. We denote by w^n the concatenation of n copies of w. The infinite word x^ω is obtained by concatenating x infinitely many times.

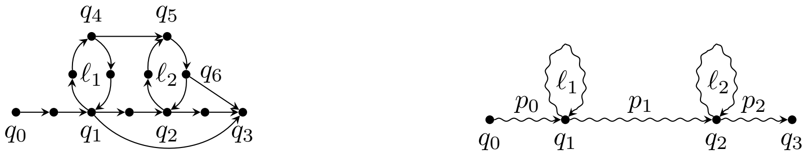
FIFO Systems.

▶ **Definition 2.1** (FIFO systems). *A FIFO system S is a tuple (Q, F, M, Δ) where Q is a finite set of control states, F is a finite set of FIFO channels, M is a finite message alphabet and $\Delta \subseteq (Q \times Q) \cup (Q \times (F \times \{!, ?\} \times M) \times Q)$ is a finite set of transitions.*

We write a transition $(q, (\mathtt{c}, ?, a), q')$ as $q \xrightarrow{\mathtt{c}?a} q'$; we similarly modify other transitions. We call q the source state and q' the target state. Transitions of the form $q \xrightarrow{\mathtt{c}?a} q'$ (resp. $q \xrightarrow{\mathtt{c}!a} q'$) denote retrieve actions (resp. send actions). Transitions of the form $q \longrightarrow q'$ do not change the channel contents but only change the control state.

The channels in F hold strings in M^*. Given two channel valuations $\mathbf{w}_1, \mathbf{w}_2 \in (M^*)^F$, we denote by $\mathbf{w}_1 \cdot \mathbf{w}_2$ the valuation obtained by concatenating the contents in $\mathbf{w}_1$ and $\mathbf{w}_2$ channel-wise. For a letter $a \in M$ and a channel $\mathtt{c} \in F$, we denote by $\mathbf{a}_\mathtt{c}$ the channel valuation that assigns a to $\mathtt{c}$ and ϵ to all other channels. The semantics of a FIFO system S is given by a transition system T_S whose set of states is $Q \times (M^*)^F$, also called configurations. Every transition $q \xrightarrow{\mathtt{c}?a} q'$ of S and channel valuation $\mathbf{w} \in (M^*)^F$ results in the transition $(q, \mathbf{a}_\mathtt{c} \cdot \mathbf{w}) \xrightarrow{\mathtt{c}?a} (q', \mathbf{w})$ in T_S. Every transition $q \xrightarrow{\mathtt{c}!a} q'$ of S and channel valuation $\mathbf{w} \in (M^*)^F$ results in the transition $(q, \mathbf{w}) \xrightarrow{\mathtt{c}!a} (q', \mathbf{w} \cdot \mathbf{a}_\mathtt{c})$ in T_S. Intuitively, the transition $q \xrightarrow{\mathtt{c}?a} q'$ (resp. $q \xrightarrow{\mathtt{c}!a} q'$) retrieves the letter a from the front of the channel $\mathtt{c}$ (resp. sends the letter a to the back of the channel $\mathtt{c}$). A run of S is a (finite or infinite) sequence of configurations $(q_0, \mathbf{w}_0)(q_1, \mathbf{w}_1) \cdots$ such that for every $i \geq 0$, there is a transition t_i such that $(q_i, \mathbf{w}_i) \xrightarrow{t_i} (q_{i+1}, \mathbf{w}_{i+1})$.

▶ **Example 2.2.** Let us present a (distributed) FIFO system (from [30]) with three processes P, Q, R that communicate through four FIFO channels pq, qp, pr, rq. Processes are extended finite automata where transitions are labeled by sending or receiving operations with FIFO channels and, for example, channel pq is an unidirectional FIFO channel from process P to process Q. From this distributed FIFO system, we get a FIFO system as given in Definition 2.1 by product construction. The control states of the product FIFO system are triples, containing control states of processes P, Q, R. The product FIFO system can go from one control state to another if one of the processes goes from a control state to another and the other two processes remain in their states. For example, the product system has the transition $(q_1, q_2, q_3) \xrightarrow{\mathtt{pq}!a_1} (q_1', q_2, q_3)$, if process P has the transition $q_1 \xrightarrow{\mathtt{pq}!a_1} q_1'$.



(a) Flat FIFO system.

(b) Path schema denoted by $p_0(\ell_1)^*p_1(l_2)^*p_2$.

Figure 2 Example flat FIFO system and path schema.

For analyzing the running time of algorithms, we assume the size of a system to be the number of bits needed to specify a system (and source/target configurations if necessary) using a reasonable encoding. Let us begin to present the reachability problems that we tackle in this paper.

▶ Problem (Reachability). *Given:* A FIFO system S and two configurations $(q_0, \mathbf{w}_0)$ and $(q, \mathbf{w})$. *Question:* Is there a run starting from $(q_0, \mathbf{w}_0)$ and ending at $(q, \mathbf{w})$?

▶ Problem (Control-state reachability). *Given:* A FIFO system S, a configuration $(q_0, \mathbf{w}_0)$ and a control-state q. *Question:* Is there a channel valuation $\mathbf{w}$ such that $(q, \mathbf{w})$ is reachable from $(q_0, \mathbf{w}_0)$?

It is folklore that reachability and control-state reachability are undecidable for machines operating on FIFO channels.

Flat systems. For a FIFO system $S = (Q, F, M, \Delta)$, its *system graph* G_S is a directed graph whose set of vertices is Q. There is a directed edge from q to q' if there is some transition $q \xrightarrow{\mathsf{c}?a} q'$ or $q \xrightarrow{\mathsf{c}!a} q'$ for some channel c and some letter a, or there is a transition $q \longrightarrow q'$. We say that S is *flat* if in G_S, every vertex is in at most one directed cycle. Figure 2a shows a flat FIFO system.

We call a FIFO system $S = (Q, F, M, \Delta)$ a *path segment* from state q_0 to state q_r if $Q = \{q_0, \ldots, q_r\}$, $\Delta = \{t_1, \ldots, t_r\}$ and for every $i \in \{1, \ldots, r\}$, q_{i-1} is the source of t_i and q_i is its target. We call a FIFO system $S = (Q, F, M, \Delta)$ an *elementary loop* on q_0 if $Q = \{q_0, \ldots, q_r\}$, $\Delta = \{t_1, \ldots, t_{r+1}\}$ and for each $i \in \{1, \ldots, r+1\}$, t_i has source q_{i-1} and target $q_{i \bmod (r+1)}$. We call $t_1 \cdots t_{r+1}$ the label of the loop. A *path schema* is a flat FIFO system comprising of a sequence $p_0\ell_1p_1\ell_2p_2 \cdots l_rp_r$, where $p_0, \ldots, p_r$ are path segments and $\ell_1, \ldots, \ell_r$ are elementary loops. There are states $q_0, q_1, \ldots, q_{r+1}$ such that p_0 is a path segment from q_0 to q_1 and for every $i \in \{1, \ldots, r\}$, p_i is a path segment from q_i to q_{i+1} and ℓ_i is an elementary loop on q_i. Except q_i, none of the other states in ℓ_i appear in other path segments or elementary loops. To emphasize that $\ell_1, \ldots, \ell_r$ are elementary loops, we denote the path schema as $p_0(\ell_1)^*p_1 \cdots (\ell_r)^*p_r$. We use the term elementary loop to distinguish them from loops in general, which may have some states appearing more than once. All loops in flat FIFO systems are elementary. Figure 2b shows a path schema, where wavy lines indicate long path segments or elementary loops that may have many intermediate states and transitions. This path schema is obtained from the flat FIFO system of Figure 2a by removing the transitions from q_1 to q_3, q_4 to q_5 and q_6 to q_3.

▶ Remark 2.3 (Fig. 1). Each process P, Q, R is flat and the cartesian product of the three automata is almost flat except on one state: there are two loops, one sending y in channel pq and another one retrieving y from channel pq.

Notations and definitions. For any sequence σ of transitions of a FIFO system and channel $\mathsf{c} \in F$, we denote by y^σ_c (resp. x^σ_c) the sequence of letters sent to (resp. retrieved from) the channel c by σ. For a configuration $(q, \mathbf{w})$, let $\mathbf{w}(\mathsf{c})$ denote the contents of channel c.

Equations on words. We recall some classical results reasoning about words and prove of one of them, to be used later. Proofs of this and a few other results are omitted. All the proofs can be found in the full version of this paper, which is on HAL with the same title. The well-known Levi's Lemma says that the words $u, v \in \Sigma^*$ that are solutions of the equation $uv = vu$ satisfy $u, v \in z^*$ where z is a primitive word. The solutions of the equation $uv = vw$ satisfy $u = xy, w = yx, v = (xy)^n x$, for some words x, y and some integer $n \geq 0$. The following lemma is used in [25] for exactly the same purpose as here.

▶ **Lemma 2.4.** *Consider three finite words $x, y \in \Sigma^+$ and $w \in \Sigma^*$. The equation $x^\omega = wy^\omega$ holds iff there exists a primitive word $z \neq \epsilon$ and two words x', x'' such that $x = x'x''$, $x''x' \in z^*$, $w \in x^*x'$ and $y \in z^*$.*

3 Complexity of Reachability Properties for Flat FIFO Systems

In this section, we give complexity bounds for the reachability problem for flat FIFO systems. We also establish the complexity of other related problems, viz. repeated control state reachability, termination, boundedness, channel boundedness and letter channel boundedness. We use the algorithm for repeated control state reachability as a subroutine for solving termination and boundedness. For channel boundedness and letter channel boundedness, we use another argument based on integer linear programming.

In [21], Esparza, Ganty, and Majumdar studied the complexity of reachability for highly undecidable models (multipushdown systems) but synchronized by bounded languages in the context of bounded model-checking. In particular, they proved that control-state reachability is NP-complete for flat FIFO systems (in fact for FIFO systems controlled by a bounded language). The NP upper bound is based on a simulation of FIFO path schemas by pushdown systems. Some constraints need to be imposed on the pushdown systems to ensure the correctness of the simulation. The structure of path schemas enables these constraints to be expressed as linear constraints on integer variables and this leads to the NP upper bound.

Surprisingly, the NP upper bound in [21] is given only for the control-state reachability problem; the complexity of the reachability problem is not established in [21] while it is given for all other considered models. However, there is a simple linear reduction from reachability to control-state reachability for FIFO (and Last In First Out) systems [32]. Such reductions are not known to exist for other models like counter systems and vector addition systems.

We begin by reducing reachability to control-state reachability (personal communication from Grégoire Sutre [32]) for (general and flat) FIFO systems.

▶ **Proposition 3.1** ([32]). *Reachability reduces (with a linear reduction) to control-state reachability, for general FIFO systems and for flat FIFO systems.*

▶ Remark 3.2. Control-state reachability is reducible to reachability for general FIFO systems. Suppose $\Sigma = \{a_1, \ldots, a_d\}$ and there are $\mathtt{p}$ channels. Using the same notations as in the previous proof, from A and q, one constructs the system $B_{A,q}$ as follows: one adds, to A, $d \times \mathtt{p}$ self loops $\ell_{i,j}$, each labeled by $\mathtt{j}?a_i$, for $i \in \{1, .., d\}$ and $j \in \{1, \ldots, \mathtt{p}\}$, all from and to the control-state q. We infer that q is reachable in A if and only if (by definition) there exists $\mathbf{w}$ such that $(q, \mathbf{w})$ is reachable in A if and only if (q, ϵ) is reachable in $B_{A,q}$. Here, (q, ϵ) denotes the configuration where q is the control state and all channels are empty. Note that $B_{A,q}$ is not necessarily flat, even if A is flat.

It is proved in [21, Theorem 7] that control state reachability is in NP for flat FIFO systems. Combining this with Proposition 3.1, we immediately deduce:

▶ **Corollary 3.3.** *Reachability is in* Np *for flat FIFO systems.*

Now we define problems concerned with infinite behaviors.

▶ Problem (Repeated reachability). *Given:* A FIFO system S, two configurations $(q_0, \mathbf{w}_0)$ and $(q, \mathbf{w})$. *Question:* Is there an infinite run from $(q_0, \mathbf{w}_0)$ such that $(q, \mathbf{w})$ occurs infinitely often along this run?

▶ Problem (Cyclicity). *Given:* A FIFO system S and a configuration $(q, \mathbf{w})$. *Question:* Is $(q, \mathbf{w})$ reachable (by a non-empty run) from $(q, \mathbf{w})$?

▶ Problem (Repeated control-state reachability). *Given:* A FIFO system S, a configuration $(q_0, \mathbf{w}_0)$ and a control-state q. *Question:* Is there an infinite run from $(q_0, \mathbf{w}_0)$ such that q occurs infinitely often along this run?

We can easily obtain an Np upper bound for repeated reachability in flat FIFO systems. A non-deterministic Turing machine first uses the previous algorithm for reachability (Corollary 3.3) to verify that $(q, \mathbf{w})$ is reachable from $(q_0, \mathbf{w}_0)$. Then the same algorithm is used again to verify that $(q, \mathbf{w})$ is reachable from $(q, \mathbf{w})$ (i.e. cyclic).

▶ **Corollary 3.4.** *Repeated reachability is in* Np *for flat FIFO systems.*

Let us recall that the cyclicity property is Expspace-complete for Petri nets [8, 20] while structural cyclicity (every configuration is cyclic) is in Ptime. Let us show that one may decide the cyclicity property for flat FIFO systems in linear time.

▶ **Lemma 3.5.** *In a flat FIFO system, a configuration $(q, \mathbf{w})$ is reachable from $(q, \mathbf{w})$ iff there is an elementary loop labeled by σ, such that $(q, \mathbf{w}) \xrightarrow{\sigma} (q, \mathbf{w})$.*

To decide whether $(q, \mathbf{w}) \xrightarrow{*} (q, \mathbf{w})$, one tests whether $(q, \mathbf{w}) \xrightarrow{\sigma} (q, \mathbf{w})$ for some elementary loop σ in the flat FIFO system. Since the FIFO system is flat, q can be in at most one loop, so only one loop need to be tested. This gives a linear time algorithm for deciding cyclicity.

▶ **Corollary 3.6.** *Testing cyclicity can be done in linear time for flat FIFO systems.*

We are now going to show an NP upper bound for repeated control state reachability.

Let a loop be labeled with σ. Recall that for each channel c, we denote by x_{c}^{σ} (resp. y_{c}^{σ}) the projection of σ to letters retrieved from (resp. sent to) the channel c. Let us write σ_{c} for the projection of σ on channel c.

▶ Remark 3.7. The loop labeled by σ is infinitely iterable from $(q, \mathbf{w})$ iff σ_{c} is infinitely iterable from $(q, \mathbf{w}(\mathsf{c}))$, for every channel c. If σ is infinitely iterable from $(q, \mathbf{w})$ then each projection σ_{c} is also infinitely iterable from $(q, \mathbf{w}(\mathsf{c}))$. Conversely, suppose σ_{c} is infinitely iterable from $(q, \mathbf{w}(\mathsf{c}))$, for every channel c. For all $\mathsf{c} \neq \mathsf{c}'$, the actions of σ_{c} and $\sigma_{\mathsf{c}'}$ are on different channels and hence independent of each other. Since σ is a shuffle of $\{\sigma_{\mathsf{c}} \mid \mathsf{c} \in F\}$, we deduce that σ is infinitely iterable from $(q, \mathbf{w})$.

We now give a characterization for a loop to be infinitely iterable.

▶ **Lemma 3.8.** *Suppose an elementary loop is on a control state q and is labeled by σ. It is infinitely iterable starting from the configuration $(q, \mathbf{w})$ iff for every channel c, $x_{\mathsf{c}}^{\sigma} = \epsilon$ or the following three conditions are true: σ is fireable at least once from $(q, \mathbf{w})$, $(x_{\mathsf{c}}^{\sigma})^{\omega} = \mathbf{w}(\mathsf{c}) \cdot (y_{\mathsf{c}}^{\sigma})^{\omega}$ and $|x_{\mathsf{c}}^{\sigma}| \leq |y_{\mathsf{c}}^{\sigma}|$.*

Proof. Let ℓ be an elementary loop on a control state q and labeled by σ. If σ is infinitely iterable starting from the configuration $(q, \mathbf{w})$ then for every channel c, one has $|x_{\mathsf{c}}| \leq |y_{\mathsf{c}}|$. Otherwise, $|x_{\mathsf{c}}| > |y_{\mathsf{c}}|$ (the number of letters retrieved is more than the number of letters sent in each iteration), so the size of the channel content reduces with each iteration, so there is a bound on the number of possible iterations. Since σ is infinitely iterable from $(q, \mathbf{w})$, the inequation $(x_{\mathsf{c}}^{\sigma})^n \leq \mathbf{w}(\mathsf{c}) \cdot (y_{\mathsf{c}}^{\sigma})^n$ must hold for all $n \geq 0$ (here, $\leq$ denotes the prefix relation). If $x_{\mathsf{c}} \neq \epsilon$, we may go at the limit and we obtain $(x_{\mathsf{c}}^{\sigma})^{\omega} \leq \mathbf{w}(\mathsf{c}) \cdot (y_{\mathsf{c}}^{\sigma})^{\omega}$.

Finally, σ is fireable at least once from $(q, \mathbf{w})$ since it is fireable infinitely from $(q, \mathbf{w})$.

Now conversely, suppose that for every channel c, $x_{\mathsf{c}}^{\sigma} = \epsilon$ or the following three conditions are true: σ is fireable at least once from $(q, \mathbf{w})$, $(x_{\mathsf{c}}^{\sigma})^{\omega} = \mathbf{w}(\mathsf{c}) \cdot (y_{\mathsf{c}}^{\sigma})^{\omega}$ and $|x_{\mathsf{c}}^{\sigma}| \leq |y_{\mathsf{c}}^{\sigma}|$. For the rest of this proof, we fix a channel c and write $x_{\mathsf{c}}^{\sigma}, y_{\mathsf{c}}^{\sigma}, \mathbf{w}(\mathsf{c})$ as x, y, w to simplify the notation.

If $x = \epsilon$ then σ is infinitely iterable because it doesn't retrieve anything. So assume that $x \neq \epsilon$. We have $x^{\omega} = wy^{\omega}$ from the hypothesis. We infer from Lemma 2.4 that there is a primitive word $z \neq \epsilon$ and words x', x'' such that $x = x'x''$, $x''x' \in z^*$, $w \in x^*x'$ and $y \in z^*$. Suppose $x''x' = z^j$ and $y = z^k$. Since $|y| \geq |x| = |x''x'|$, we have $k \geq j$. Let us prove the following monotonicity property: for all $n \geq 0$, σ is fireable from any channel content wz^n and the resulting channel content is $wz^{n+(k-j)}$ (this will imply that for all $m \geq 1$, $w \xrightarrow{\sigma^m} wz^{m\times(k-j)}$, hence that σ is infinitely iterable). We prove the monotonicity property by induction on n.

For the base case $n = 0$, we need to prove that $w \xrightarrow{\sigma} wz^{k-j}$. By hypothesis, σ is fireable at least once from w, hence $w \xrightarrow{\sigma} w'$ for some w'. We have $w' = x^{-1}wy = x^{-1}x^r x' z^k$ for some $r \in \mathbb{N}$. Since $k \geq j$, we have $w' = x^{-1}x^r x' z^j z^{k-j} = x^{-1}x^r x'(x''x')z^{k-j} = x^{-1}x^r(x'x'')x'z^{k-j} = x^{-1}x^{r+1}x'z^{k-j} = x^r x' z^{k-j} = wz^{k-j}$.

For the induction step, we have to show that σ is fireable from channel content wz^{n+1} and the resulting channel content is $wz^{n+1+(k-j)}$. From induction hypothesis, we know that σ is fireable from channel content wz^n. Since $y = z^k$, the channel content after firing a prefix σ_1 of σ is $x_1^{-1}wz^n z^s z_1$, where x_1 is some prefix of x, $s \in \mathbb{N}$ and z_1 is some prefix of z. By induction on $|\sigma_1|$, we can verify that σ_1 can be fired from wz^{n+1} and results in $x_1^{-1}wz^{n+1}z^s z_1$. Hence, σ can be fired from wz^{n+1} and results in $x^{-1}wz^{n+1}y = x^{-1}x^r x' z^{n+1} z^k = x^{-1}x^r x' z^j z^{n+1+k-j} = x^{-1}x^r x' x'' x' z^{n+1+k-j} = x^{-1}x^{r+1}x'z^{n+1+k-j} = wz^{n+1+k-j}$. This completes the induction step and hence proves the monotonicity property.

Hence σ is infinitely iterable. ◀

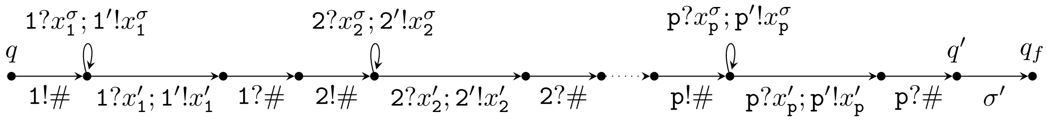
The proof of Lemma 3.8 provides a complete characterization of the contents of a FIFO channel when a loop is infinitely iterable. One may observe that the channel acts like a counter (of the number of occurrences of z).

▶ **Corollary 3.9.** *With the previous notations, the set of words in channel* c *that occur in control-state* q *is the regular periodic language* $\mathbf{w}(\mathsf{c}) \cdot [z_{\mathsf{c}}^{k-j}]^*$*, when the elementary loop containing* q *is iterated arbitrarily many times.*

▶ Remark 3.10. One may find other similar results on infinitely iterable loops in many papers [22, 29, 6, 7, 25]. Our Lemma 3.8 is the same as [25, Proposition 5.1] except that it (easily) extends it to systems with *multiple* channels and also provides the converse. Lemma 3.8 simplifies and improves Proposition 5.4. in [7] that used the equivalent but more complex notion of *inc-repeating sequence.* Also, the results in [7] don't give the simple representation of the regular periodic language.

▶ **Lemma 3.11.** *The repeated control state reachability problem is in* NP *for flat FIFO systems.*

Proof. We describe an NP algorithm. Suppose S is the given flat FIFO system and the control state q is to be reached repeatedly. Suppose q is in a loop labeled with σ. The algorithm first verifies that for every channel $\mathtt{c}$, $|x^\sigma_\mathtt{c}| \leq |y^\sigma_\mathtt{c}|$ – if this condition is violated, the answer is no. From Lemma 3.8, it is enough to verify that we can reach a configuration $(q, \mathbf{w})$ such that σ can be fired at least once from $(q, \mathbf{w})$ and for every channel $\mathtt{c}$ for which $x^\sigma_\mathtt{c} \neq \epsilon$, we have $(x^\sigma_\mathtt{c})^\omega = \mathbf{w}(\mathtt{c}) \cdot (y^\sigma_\mathtt{c})^\omega$. Since the case of $x^\sigma_\mathtt{c} = \epsilon$ can be handled easily, we assume in the rest of this proof that $x^\sigma_\mathtt{c} \neq \epsilon$ for every $\mathtt{c}$. For verifying that $(x^\sigma_\mathtt{c})^\omega = \mathbf{w}(\mathtt{c}) \cdot (y^\sigma_\mathtt{c})^\omega$, the algorithm depends on Lemma 2.4: the algorithm guesses $x'_\mathtt{c}, x''_\mathtt{c}, z_\mathtt{c} \in M^*$ such that $x^\sigma_\mathtt{c} = x'_\mathtt{c} x''_\mathtt{c}$ and $x''_\mathtt{c} x'_\mathtt{c}, y^\sigma_\mathtt{c} \in z^*_\mathtt{c}$. We have $|x'_\mathtt{c}|, |x''_\mathtt{c}| \leq |x^\sigma_\mathtt{c}|$ and $|z_\mathtt{c}| \leq |y^\sigma_\mathtt{c}|$ so the guessed strings are of size bounded by the size of the input. It remains to verify that we can reach a configuration $(q, \mathbf{w})$ such that for every channel $\mathtt{c}$, $\mathbf{w}(\mathtt{c}) \in (x^\sigma_\mathtt{c})^* x'_\mathtt{c}$ and σ can be fired at least once from $(q, \mathbf{w})$. For accomplishing these two tasks, we add a channel $\mathtt{c}'$ for every channel $\mathtt{c}$ in the FIFO system S. The following gadgets are appended to the control state q, assuming that there are $\mathtt{p}$ channels and $\#$ is a special letter not in the channel alphabet M. We denote by σ' the sequence of transitions obtained from σ by replacing every channel $\mathtt{c}$ by $\mathtt{c}'$. A transition labeled with $\mathtt{c}?x^\sigma_\mathtt{c}; \mathtt{c}'!x^\sigma_\mathtt{c}$ is to be understood as a sequence of transitions whose effect is to retrieve $x^\sigma_\mathtt{c}$ from channel $\mathtt{c}$ and send $x^\sigma_\mathtt{c}$ to channel $\mathtt{c}'$.


$1?x^\sigma_1; 1'!x^\sigma_1$
$2?x^\sigma_2; 2'!x^\sigma_2$
$\mathtt{p}?x^\sigma_\mathtt{p}; \mathtt{p}'!x^\sigma_\mathtt{p}$
q
q'
q_f
$1!\#$
$1?x'_1; 1'!x'_1$
$1?\#$
$2!\#$
$2?x'_2; 2'!x'_2$
$2?\#$
$\mathtt{p}!\#$
$\mathtt{p}?x'_\mathtt{p}; \mathtt{p}'!x'_\mathtt{p}$
$\mathtt{p}?\#$
σ'

Finally our algorithm runs the NP algorithm to check that the control state q_f is reachable. We claim that the control state q can be visited infinitely often iff our algorithm accepts. Suppose q can be visited infinitely often. So the loop containing q can be iterated infinitely often. Hence from Lemma 3.8, we infer that S can reach a configuration $(q, \mathbf{w})$ such that σ can be fired at least once and for every channel $\mathtt{c}$, $|x^\sigma_\mathtt{c}| \leq |y^\sigma_\mathtt{c}|$ and $(x^\sigma_\mathtt{c})^\omega = \mathbf{w}(\mathtt{c}) \cdot (y^\sigma_\mathtt{c})^\omega$. From Lemma 2.4, there exist $x'_\mathtt{c}, x''_\mathtt{c}, z_\mathtt{c} \in M^*$ such that $x^\sigma_\mathtt{c} = x'_\mathtt{c} x''_\mathtt{c}$, $\mathbf{w}(\mathtt{c}) \in (x^\sigma_\mathtt{c})^* x'_\mathtt{c}$ and $x''_\mathtt{c} x'_\mathtt{c}, y^\sigma_\mathtt{c} \in z^*_\mathtt{c}$. Our algorithm can guess exactly these words $x'_\mathtt{c}, x''_\mathtt{c}, z_\mathtt{c}$. It is easy to verify that from the configuration $(q, \mathbf{w})$, the configuration $(q', \mathbf{w}')$ can be reached, where $\mathbf{w}'(\mathtt{c}') = \mathbf{w}(\mathtt{c})$ for every $\mathtt{c}$. Since σ can be fired from $(q, \mathbf{w})$, σ' can be fired from $(q', \mathbf{w}')$ to reach q_f. So our algorithm accepts.

Conversely, suppose our algorithm accepts. Hence the control state q_f is reachable. By construction, we can verify that the run reaching the control state q_f has to visit a configuration $(q, \mathbf{w})$ such that for every channel $\mathtt{c}$, $\mathbf{w}(\mathtt{c}) \in (x^\sigma_\mathtt{c})^* x'_\mathtt{c}$ and σ can be fired at least once from $(q, \mathbf{w})$. Our algorithm also verifies that $|x^\sigma_\mathtt{c}| \leq |y^\sigma_\mathtt{c}|$, $x^\sigma_\mathtt{c} = x'_\mathtt{c} x''_\mathtt{c}$ and $x''_\mathtt{c} x'_\mathtt{c}, y^\sigma_\mathtt{c} \in z^*_\mathtt{c}$. Hence, from Lemma 2.4 and Lemma 3.8, we infer that the loop containing q can be iterated infinitely often starting from the configuration $(q, \mathbf{w})$. Hence, there is a run that visits q infinitely often. ◀

Let us now introduce the non-termination and the unboundedness problems.

▶ Problem (Non-termination). *Given:* A FIFO system S and an initial configuration $(q_0, \mathbf{w}_0)$. *Question:* Is there an infinite run from $(q_0, \mathbf{w}_0)$?

▶ Problem (Unboundedness). *Given:* A FIFO system S and an initial configuration $(q_0, \mathbf{w}_0)$. *Question:* Is the set of configurations reachable from $(q_0, \mathbf{w}_0)$ infinite?

▶ **Corollary 3.12.** *For flat FIFO systems, the non-termination and unboundedness problems are in* NP.

For a word w and a letter a, $|w|_a$ denotes the number of occurrences of a in w. For a FIFO system, we say that a letter a is unbounded in channel c if for every number B, there exists a reachable configuration $(q, \mathbf{w})$ with $|\mathbf{w}(\mathsf{c})|_a \geq B$. A channel c is unbounded if at least one letter a is unbounded in c.

▶ Problem (Channel-unboundedness). *Given:* A FIFO system S, an initial configuration $(q_0, \mathbf{w}_0)$ and a channel c. *Question:* Is the channel c unbounded from $(q_0, \mathbf{w}_0)$?

▶ Problem (Letter-channel-unboundedness). *Given:* A FIFO system S, an initial configuration $(q_0, \mathbf{w}_0)$, a channel c and a letter a. *Question:* Is the letter a unbounded in channel c from $(q_0, \mathbf{w}_0)$?

Now we give an NP upper bound for letter channel unboundedness in flat FIFO systems. We use the following two results in our proof.

▶ **Theorem 3.13** ([21, Theorem 3, Theorem 7]). *Let $S = p_0(\ell_1)^* p_1 \cdots (\ell_r)^* p_r$ be a FIFO path schema. We can compute in polynomial time an existential Presburger formula $\phi(x_1, \ldots, x_r)$ satisfying the following property: there is a run of S in which the loop ℓ_i is iterated exactly n_i times for every $i \in \{1, \ldots, r\}$ iff $\phi(n_1, \ldots, n_r)$ is true.*

For vectors $\mathbf{k}, \mathbf{x}$ and matrix $\mathbf{A}$, the expression $\mathbf{k} \cdot \mathbf{x}$ denotes the dot product and the expression $\mathbf{Ax}$ denotes the matrix product.

▶ **Theorem 3.14** ([31, Lemma 3]). *Suppose $\mathbf{A}$ is an integer matrix and $\mathbf{k}, \mathbf{b}$ are integer vectors satisfying the following property: for every $B \in \mathbb{N}$, there exists a vector $\mathbf{x}$ of* rational numbers *such that $\mathbf{Ax} \geq \mathbf{b}$ and $\mathbf{k} \cdot \mathbf{x} \geq B$. If there is an integer vector $\mathbf{x}$ such that $\mathbf{Ax} \geq \mathbf{b}$, then for every $B \in \mathbb{N}$, there exists an* integer *vector $\mathbf{x}$ such that $\mathbf{Ax} \geq \mathbf{b}$ and $\mathbf{k} \cdot \mathbf{x} \geq B$.*

▶ **Theorem 3.15.** *Given a flat FIFO system, a letter a and channel c, the problem of checking whether a is unbounded in c is in NP.*

Proof. The letter a is unbounded in c iff there exists a control state q such that for every number B, there is a reachable configuration with control state q and at least B occurrences of a in channel c (this follows from definitions since there are only finitely many control states). A non-deterministic polynomial time Turing machine begins by guessing a control state q. If there are r loops in the path schema ending at q, the Turing machine computes an existential Presburger formula $\phi(x_1, \ldots, x_r)$ satisfying the following property: $\phi(n_1, \ldots, n_r)$ is true iff there is a run ending at q in which loop i is iterated n_i times for every $i \in \{1, \ldots, r\}$. Such a formula can be computed in polynomial time (Theorem 3.13). Let k_i be the number of occurrences of the letter a sent to channel c by one iteration of the i^{th} loop (k_i would be negative if a is retrieved instead). If loop i is iterated n_i times for every i in a run, then at the end of the run there are $k_1 n_1 + \cdots + k_r n_r$ occurrences of the letter a in channel c. To check that a is unbounded in channel c, we have to verify that there are tuples $\langle n_1, \ldots, n_r \rangle$ such that $\phi(n_1, \ldots, n_r)$ is true and $k_1 n_1 + \cdots + k_r n_r$ is arbitrarily large. This is easier to do if there are no disjunctions in the formula $\phi(x_1, \ldots, x_r)$. If there are any sub-formulas with disjunctions, the Turing machine non-deterministically chooses one of the disjuncts and drops the other one. This is continued till all disjuncts are discarded. This results in a conjunction of linear inequalities, say $A\mathbf{x} \geq \mathbf{b}$, where $\mathbf{x}$ is the tuple of variables $\langle x_1, \ldots, x_r \rangle$. The machine then tries to maximize $k_1 x_1 + \cdots + k_r x_r$ over rationals subject to the constraints $A\mathbf{x} \geq \mathbf{b}$. This can be done in polynomial time, since linear programming is in polynomial time. If the value $k_1 x_1 + \cdots + k_r x_r$ is unbounded above over rationals subject to the constraints $A\mathbf{x} \geq \mathbf{b}$, then the machine invokes the NP algorithm to check if the constraints $A\mathbf{x} \geq \mathbf{b}$ has a feasible solution over integers. If it does, then $k_1 x_1 + \cdots + k_r x_r$ is also unbounded above over integers (Theorem 3.14). Hence, in this case, a is unbounded in channel c. ◀

The above result also gives an NP upper bound for channel-unboundedness. We just guess a letter a and check that it is unbounded in the given channel.

We adapt the proof of NP-hardness for the control state reachability problem from [21] to prove NP hardness for reachability, repeated control state reachability, unboundedness and non-termination.

▶ **Lemma 3.16.** *For flat FIFO systems, reachability, repeated control-state reachability, non-termination, unboundedness, channel-unboundedness and letter-channel-unboundedness are NP-hard.*

Hence we deduce the main result of this Section.

▶ **Theorem 3.17** (Most properties are NP-complete)**.** *For flat FIFO systems, reachability, repeated reachability, repeated control-state reachability, termination, boundedness, channel-boundedness and letter-channel-boundedness are NP-complete. Cyclicity can be decided in linear time.*

4 Construction of an Equivalent Counter System

Suppose we want to model check flat FIFO systems against logics in which atomic formulas are of the form $\#_{\mathsf{c}}^{a} \geq k$, which means there are at least k occurrences of the letter a in channel c. There is no easy way of designing an algorithm for this model checking problem based on the construction in [21], even though we solved reachability and related problems in previous sections using that construction. That construction is based on simulating FIFO systems using automata that have multiple reading heads on an input tape. The channel contents of the FIFO system are represented in the automaton as the sequence of letters on the tape between two reading heads. There is no way in the automaton to access the tape contents between two heads, and hence no way to check the number of occurrences of a specific letter in a channel. CQDDs introduced in [7] represent the entire set of reachable states and they are also not suitable for model checking. To overcome this problem, we introduce here a counter system to simulate flat FIFO systems. This has the additional advantage of being amenable to analysis using existing tools on counter machines.

Counter systems are finite state automata augmented with counters that can store natural numbers. Let K be a finite set of counters and let *guards over* K be the set $G(K)$ of positive Boolean combinations[1] of constraints of the form $C = 0$ and $C > 0$, where $C \in K$.

▶ **Definition 4.1** (Counter systems)**.** *A counter system S is a tuple $\langle Q, K, \Delta\rangle$ where Q is a finite set of control states and $\Delta \subseteq Q \times G(K) \times \{-1, 0, 1\}^K \times Q$ is a finite set of transitions.*

We may add one or two labeling functions to the tuple $\langle Q, K, \Delta\rangle$ to denote labeled counter systems. The semantics of a counter system is a transition system with set of states $Q \times \mathbb{N}^K$, called *configurations of the counter system*. A counter valuation $\nu \in \mathbb{N}^K$ *satisfies* a guard $C = 0$ (resp. $C > 0$) if $\nu(C) = 0$ (resp. $\nu(C) > 0$), written as $\nu \models C = 0$ (resp. $\nu \models C > 0$). The satisfaction relation is extended to Boolean combinations in the standard way. For every transition $\delta = q \xrightarrow[g]{\mathbf{u}} q'$ in the counter system, we have transitions $(q, \nu_1) \xrightarrow{\delta} (q', \nu_2)$ in the associated transition system for every ν_1 such that $\nu_1 \models g$ and $\nu_2 = \nu_1 + \mathbf{u}$ (addition of

[1] In the literature, counter systems can have more complicated guards, such as Presburger constraints. For our purposes, this restricted version suffices.

vectors is done component-wise). We write a transition $(q, C_2 = 0, \langle 1, 0\rangle, q')$ as $q \xrightarrow[C_2=0]{C_1^{++}} q'$, denoting addition of 1 to C_1 by C_1^{++}. We denote by $\longrightarrow$ the union $\cup_{\delta\in\Delta} \xrightarrow{\delta}$. A *run* of the counter system is a finite or infinite sequence $(q_0, \nu_0) \longrightarrow (q_1, \nu_1) \longrightarrow \cdots$ of configurations, where each pair of consecutive configurations is in the transition relation.

We assume for convenience that the message alphabet M of a FIFO system is the disjoint union of $M_1, \ldots, M_{\mathtt{p}}$, where $M_{\mathtt{c}}$ is the alphabet for channel $\mathtt{c}$. In the following, let $S = (Q, F, M, \Delta)$ be a flat FIFO system, where the set of channels $F = \{1, \ldots, \mathtt{p}\}$ and the set of transitions $\Delta = \{t_1, \ldots, t_r\}$.

The *counting abstraction system* corresponding to S is a labeled counter system $S_{\text{count}} = (Q, K, \Delta_{\text{count}}, \psi, T)$, where $(Q, K, \Delta_{\text{count}})$ is a counter system and ψ, T are labeling functions. The set of *counters* K is in bijection with $M \times \Delta$ and a counter will be denoted $c_{a,t}$ or shortly (a, t), for $a \in M$ and $t \in \Delta$. The set Δ_{count} of *transitions* of S_{count} and the labeling functions $\psi : \Delta_{\text{count}} \to (M \times \Delta) \cup \{\tau\}$ and $T : \Delta_{\text{count}} \to \Delta$ are defined as follows: for every transition $t \in \Delta$, one adds the following transitions in Δ_{count} :

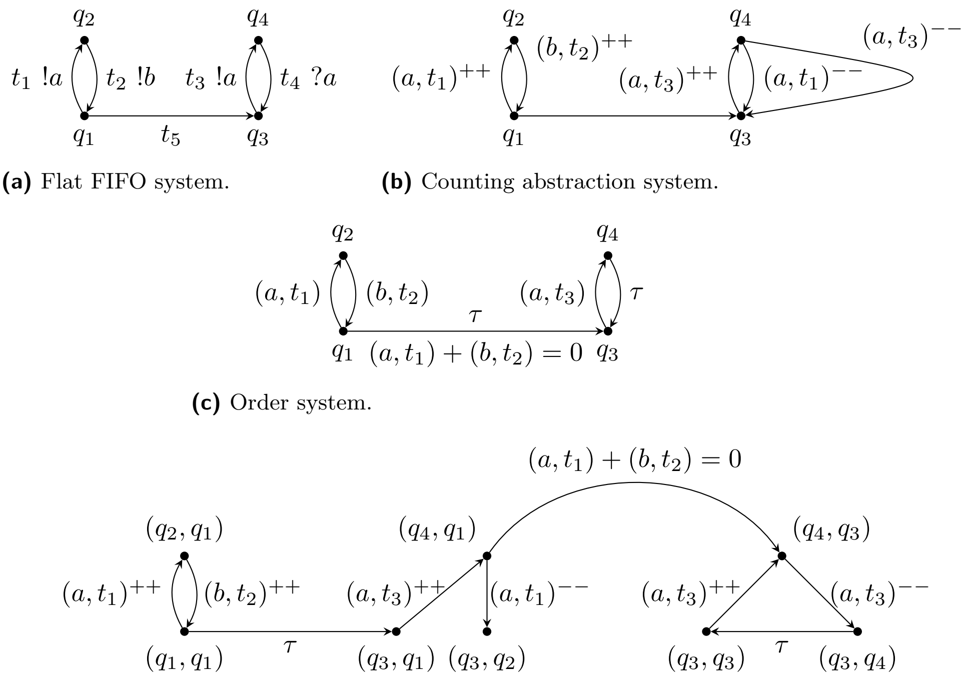
- If t *sends* a message, $t = q_1 \xrightarrow{\mathtt{c}!a} q_2$, then the transition $t_{\text{count}} = q_1 \xrightarrow{(a,t)^{++}} q_2$ is added to Δ_{count} ; we define $\psi(t_{\text{count}}) = \tau$ and $T(t_{\text{count}}) = t$.
- If $t = q_1 \longrightarrow q_2$ doesn't change any channel content, then the transition $t_{\text{count}} = q_1 \longrightarrow q_2$ is added to Δ_{count} ; we define $\psi(t_{\text{count}}) = \tau$ and $T(t_{\text{count}}) = t$.
- If t *receives* a message, $t = q_1 \xrightarrow{\mathtt{c}?a} q_2$, then the set of transitions A_t is added to Δ_{count} with $A_t = \{\delta_{a,t'} = q_1 \xrightarrow[(a,t')>0]{(a,t')^{--}} q_2 \mid t' \text{ sends } a \text{ to channel } \mathtt{c}\}$. We define $\psi(\delta_{a,t'}) = (a, t')$ and $T(\delta_{a,t'}) = t$, for all $\delta_{a,t'} \in A_t$.

The function ψ above will be used for synchronization with other counter systems later and T will be used to match the traces of this counter system with those of the original flat FIFO system. In figures, we do not show the labels given by ψ and T. They can be easily determined. For a transition $\delta_{a,t'} \in \Delta_{\text{count}}$, it decrements the counter (a, t') and $\psi(\delta_{a,t'}) = (a, t')$. Transitions that don't decrement any counter are mapped to τ by ψ.

▶ **Example 4.2.** Figure 3a shows a flat FIFO system and Fig. 3b shows its counting abstraction system.

The idea behind the counting abstraction system is to *ignore the order of letters* stored in the channels and use counters to remember only the number of occurrences of each letter. If a transition t sends letter a, the corresponding transition in the counting abstraction system increments the counter (a, t). If a transition t retrieves a letter a, the retrieved letter would have been produced by some earlier transition t'; the corresponding transition in the counting abstraction system will decrement the counter (a, t'). The counting abstraction system doesn't exactly simulate the flat FIFO system. For example, if the transition labeled $(a, t_1)^{--}$ in Fig. 3b is executed, we know that there is at least one occurrence of the letter a in the channel, since the counter (a, t_1) is greater than zero at the beginning of the transition. However, it is not clear that the letter a is at the front of the channel; there might be an occurrence of the letter b at the front. This condition can't be tested using the counting abstraction system. We use other counter systems to maintain the order of letters.

The *order system for channel* $\mathtt{c}$ is a labeled counter system $S^{\mathtt{c}}_{\text{order}} = (Q, K, \Delta^{\mathtt{c}}_{\text{order}}, \psi^{\mathtt{c}})$, where $(Q, K, \Delta^{\mathtt{c}}_{\text{order}})$ is a counter system and $\psi^{\mathtt{c}}$ is a labeling function. The set of control states Q and the set of counters K are the same as in the counting abstraction system. The set $\Delta^{\mathtt{c}}_{\text{order}}$ of *transitions* of $S^{\mathtt{c}}_{\text{order}}$ and the *labeling function* $\psi^{\mathtt{c}} : \Delta^{\mathtt{c}}_{\text{order}} \to (M \times \Delta) \cup \{\tau\}$ are defined as follows: for every $t \in \Delta$, one adds the following transitions in $\Delta^{\mathtt{c}}_{\text{order}}$:



(a) Flat FIFO system.

(b) Counting abstraction system.

(c) Order system.

(d) Synchronized counter system.

Figure 3 An example flat FIFO system and the equivalent counter system.

- If $t = q_1 \xrightarrow{\mathsf{c}!a} q_2$, one adds to $\Delta^{\mathsf{c}}_{\text{order}}$ the transition $t' = q_1 \to q_2$ and $\psi^{\mathsf{c}}(t') = (a,t)$.
- If $t = q_1 \xrightarrow{x} q_2$ where x doesn't contain a sending operation (of a letter) to channel c, one adds to $\Delta^{\mathsf{c}}_{\text{order}}$ the transition $t' = q_1 \to q_2$ and $\psi^{\mathsf{c}}(t') = \tau$.

While adding the transitions above, if t happens to be the first transition after and outside a loop in S, we add a guard to the transition t' that we have given in the above two cases. Suppose t is the first transition after and outside a loop, and the loop is labeled by σ. We add the following guard to the transition t'.

$$\sum_{\substack{t'' \text{ occurs in } \sigma \\ a \in M}} (a,t'') = 0$$

Figure 3c shows the order system corresponding to the flat FIFO system of Fig. 3a.

We will synchronize the counting abstraction system with the order systems by rendez-vous on transition labels. Suppose the order system is in state q_2 as shown in Fig. 3c. The only transition going out from q_2 is labeled by (b,t_2), denoting the fact that the front of the channel contains b. The counting abstraction system can't execute the transition labeled with $(a,t_1)^{--}$ in this configuration, since its ψ-label is (a,t_1) and hence it can't synchronize with the order system, whose next transition is labeled with (b,t_2). The guard $(a,t_1)+(b,t_2) = 0$ in the bottom transition in Fig. 3c ensures that all occurrences of letters produced by iterations of the first loop are retrieved before those produced by the second loop.

In the following, the *label of a transition* refers to the image of that transition under the function ψ (if the transition is in the counting abstraction system) or the function ψ^{c} (if the transition is in the order system for channel c). The *synchronized counter system* $S_{\text{sync}} = S_{\text{count}} \mid\mid S^{1}_{\text{order}} \mid\mid \ldots \mid\mid S^{\mathsf{c}}_{\text{order}} \mid\mid \ldots \mid\mid S^{\mathsf{p}}_{\text{order}}$ is the synchronized (by rendez-vous) product of the *counting abstraction system* S_{count} and the *order systems* $S^{\mathtt{c}}_{\mathrm{order}}$ for all channels $\mathtt{c} \in \{1, \ldots, \mathtt{p}\}$. All counter systems share the same set of counters K and have disjoint copies of the set of control states Q, so the global control states of the synchronized counter system are tuples in $Q^{\mathtt{p}+1}$. Transitions labeled with τ need not synchronize with others. Each transition labeled (by the function ψ or $\psi^{\mathtt{c}}$ as explained above) with an element of $M \times \Delta$ should synchronize with exactly one other transition that is similarly labeled. We extend the labeling function T of S_{count} to S_{sync} as follows: if a transition t of S_{count} participates in a transition t_s of S_{sync}, then $T(t_s) = T(t)$. If no transition from S_{count} participates in t_s, then $T(t_s) = \tau$ and we call t_s a silent transition.

Since we have assumed that the channel alphabets for different channels are mutually disjoint, synchronizations can only happen between the counting abstraction system and one of the order systems. For a global control state $\overline{q} \in Q^{\mathtt{p}+1}$, $\overline{q}(0)$ denotes the local state of the counting abstraction system and $\overline{q}(\mathtt{c})$ denotes the local state of the order system for channel $\mathtt{c}$. The synchronized counter system maintains the channel contents of the flat FIFO system as explained next.

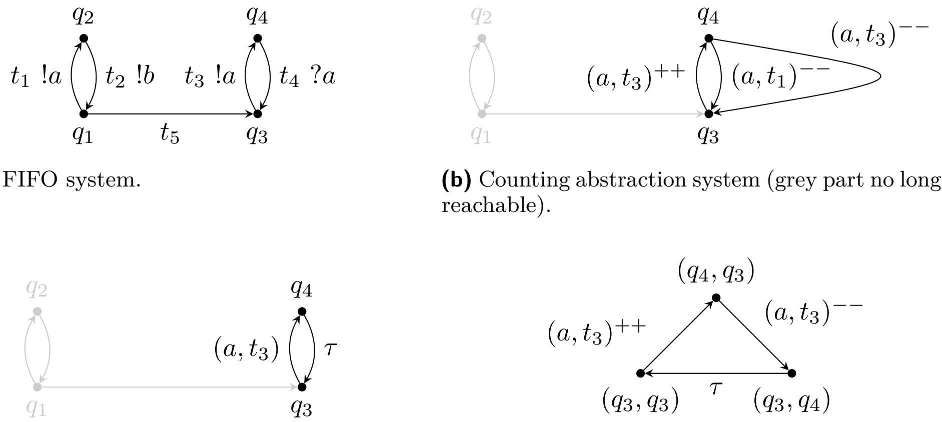
We now explain that every reachable configuration $(\overline{q}, \nu)$ of S_{sync} corresponds to a unique configuration $h(\overline{q}, \nu)$ of the original FIFO system S. The corresponding configuration of S is $(\overline{q}(0), h_1(v_1), h_2(v_2), \ldots h_{\mathtt{p}}(v_{\mathtt{p}}))$, where the words $v_{\mathtt{c}} \in \Delta^*$ and morphisms $h_{\mathtt{c}} : \Delta^* \to M^*$ are as follows. Fix a channel $\mathtt{c}$. Let $v_{\mathtt{c}} \in \Delta^*$ be a word labelling a path in S from $\overline{q}(\mathtt{c})$ to $\overline{q}(0)$ such that $\mathit{Parikh}(v_{\mathtt{c}})(t) = \nu\left((a, t)\right)$ for every transition $t \in \Delta$ that sends some letter to channel $\mathtt{c}$ (and a is the letter that is sent by t). Now, define $h_{\mathtt{c}}(t) = a$ if t sends some letter to channel $\mathtt{c}$ (and a is the letter sent) and $h_{\mathtt{c}}(t) = \epsilon$ otherwise. The word $h_c(v_{\mathtt{c}})$ is unique since S is flat and so the set of traces of S, interpreted as a language over the alphabet Δ, is included in a bounded language. Intuitively, the path $v_{\mathtt{c}}$ gives the order of letters in channel $\mathtt{c}$ and the counters give the number of occurrences of each letter.

▶ **Example 4.3.** Figure 3d shows the reachable states of the synchronized counter system for the flat FIFO system in Fig. 3a. Initially, both the counting abstraction system and the order system are in state q_1, so the global state is (q_1, q_1). Then the counting abstraction system may execute the transition labeled $(a, t_1)^{++}$ and go to state q_2 while the order system stays in state q_1, resulting in the global state (q_2, q_1). Consider the global state $\overline{q} = (q_3, q_2)$ and counter valuation ν with $\nu((a, t_1)) = 2, \nu((b, t_2)) = 3$ and $\nu((a, t_3)) = 1$. Then, for the only channel $\mathtt{c} = \mathtt{1}$, $v_{\mathtt{c}} = t_2(t_1 t_2)^2 t_5 t_3$ and $h_{\mathtt{c}}(v_{\mathtt{c}}) = b(ab)^2 a$.

A relation R between the reachable configurations of the FIFO system S and the synchronized counter system S_{sync} is a *weak bisimulation* if every pair $((q, \mathbf{w}), (\overline{q}, \nu)) \in R$ satisfies the following conditions: (1) for every transition $(q, \mathbf{w}) \xrightarrow{t} (q', \mathbf{w}')$ in S, there is a sequence σ of transitions in S_{sync} such that $T(\sigma) \in \tau^* t \tau^*$, $(\overline{q}, \nu) \xrightarrow{\sigma} (\overline{q'}, \nu')$ and $((q', \mathbf{w}'), (\overline{q'}, \nu')) \in R$, (2) for every transition $(\overline{q}, \nu) \xrightarrow{t_s} (\overline{q'}, \nu')$ in S_{sync} with $T(t_s) = \tau$, $((q, \mathbf{w}), (\overline{q'}, \nu')) \in R$ and (3) for every transition $(\overline{q}, \nu) \xrightarrow{t_s} (\overline{q'}, \nu')$ in S_{sync} with $T(t_s) = t \neq \tau$, $(q, \mathbf{w}) \xrightarrow{t} (q', \mathbf{w}')$ is a transition in S and $((q', \mathbf{w}'), (\overline{q'}, \nu')) \in R$.

▶ **Lemma 4.4.** *The relation* $\{(h((\overline{q}, \nu)), (\overline{q}, \nu)) \mid (\overline{q}, \nu)$ *is reachable in* $S_{\mathrm{sync}}\}$ *is a weak bisimulation.*

The synchronized counter system S_{sync} is not flat. E.g., there are two transitions from q_4 to q_3 in Fig. 3b. Those two states are in more than one loop, violating the condition of flatness. However, suppose a run is visiting states q_3, q_4 of the counting abstraction system and states q_3, q_4 of the order system as shown in Fig. 4 (parts of the systems that are no longer reachable are greyed out). Now the transition labeled $(a, t_1)^{--}$ can't be used and the



(a) Flat FIFO system.

(b) Counting abstraction system (grey part no longer reachable).

(c) Order system (grey part no longer reachable).

(d) Part of synchronized counter system still reachable.

Figure 4 Flattening.

run is as shown in Fig. 4d, which is a flat counter system. In general, suppose $\ell_0, \ell_1, \ldots, \ell_r$ are the loops in S. There is a flat counter system S_{flat} whose set of runs is the set of runs ρ of the synchronized transition system which satisfy the following property: in ρ, all local states of the counting abstraction system are in some loop ℓ_i and for every channel $\mathtt{c}$, all local states of the order system $S^{\mathtt{c}}_{\mathrm{order}}$ are in some loop $\ell_{\mathtt{c}}$. This is the intuition for the next result.

Let traces(S_{sync}) be the set of all runs of S_{sync}. Let S' be another counter system with set of states Q' and the same set of counters as S_{sync} and let $f : Q' \to Q$ be a function. We say that S' is a f-flattening of S_{sync} [15, Definition 6] if S' is flat and for every transition $q \xrightarrow[g]{u} q'$ of S', $f(q) \xrightarrow[g]{u} f(q')$ is a transition in S_{sync}. Further, S' is a f-trace-flattening of S_{sync} [15, Definition 8] if S' is a f-flattening of S_{sync} and traces(S_{sync}) $= f$(traces(S')).

▶ **Lemma 4.5.** *The synchronized counter system S_{sync} is trace-flattable.*

Let S_{flat} be a trace-flattening of S_{sync}. In general, the size of S_{flat} is exponential in the size of S_{sync}, which is exponential in the size of S. The weak bisimulation shown in Lemma 4.4 can be strengthened to bisimulation; see the full version for details. In theory, problems on flat FIFO systems can be solved by using tools on counter systems (bisimulation preserves CTL* and trace-flattening preserves LTL [15, Theorem1]). It remains to be seen if tools can be optimized to make verifying FIFO systems work in practice.

5 Conclusion and Perspectives

We answered the complexity of the main reachability problems for flat FIFO systems which are NP-complete as for flat counter systems. We also show how to translate a flat FIFO system into a trace-flattable counter system. This opens the way to model-check general FIFO systems by *enumerating their flat subsystems*. For example, if we construct the product of the three processes shown in Fig. 1, the resulting FIFO system is not flat. It does become flat if we remove the self loop labeled $\mathtt{pq}?y$. The resulting flat subsystem is unbounded, so it implies that the original system is also unbounded. Hence, even if the given FIFO system is not flat, some questions can often be answered by analyzing flat subsystems. This strategy has worked well for counter systems and offers hope for FIFO systems.

References

1 Parosh Aziz Abdulla, Aurore Collomb-Annichini, Ahmed Bouajjani, and Bengt Jonsson. Using Forward Reachability Analysis for Verification of Lossy Channel Systems. *Formal Methods in System Design*, 25(1):39–65, 2004. doi:10.1023/B:FORM.0000033962.51898.1a.

2 Aurore Annichini, Ahmed Bouajjani, and Mihaela Sighireanu. TReX: A Tool for Reachability Analysis of Complex Systems. In Gérard Berry, Hubert Comon, and Alain Finkel, editors, *Computer Aided Verification*, pages 368–372, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

3 Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Laure Petrucci. FAST: Fast Acceleration of Symbolic Transition systems. In Warren A. Hunt, Jr and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification (CAV'03)*, volume 2725 of *Lecture Notes in Computer Science*, pages 118–121, Boulder, Colorado, USA, July 2003. Springer. URL: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PS/FAST-cav03.ps.

4 Sébastien Bardin, Alain Finkel, Jérôme Leroux, and Philippe Schnoebelen. Flat acceleration in symbolic model checking. In Doron A. Peled and Yih-Kuen Tsay, editors, *Proceedings of the 3rd International Symposium on Automated Technology for Verification and Analysis (ATVA'05)*, volume 3707 of *Lecture Notes in Computer Science*, pages 474–488, Taipei, Taiwan, October 2005. Springer. doi:10.1007/11562948_35.

5 Bernard Boigelot. Domain-specific regular acceleration. *STTT*, 14(2):193–206, 2012. doi:10.1007/s10009-011-0206-x.

6 Bernard Boigelot, Patrice Godefroid, Bernard Willems, and Pierre Wolper. The Power of QDDs (Extended Abstract). In Pascal Van Hentenryck, editor, *Static Analysis, 4th International Symposium, SAS '97, Paris, France, September 8-10, 1997, Proceedings*, volume 1302 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 1997. doi:10.1007/BFb0032741.

7 Ahmed Bouajjani and Peter Habermehl. Symbolic Reachability Analysis of FIFO-Channel Systems with Nonregular Sets of Configurations. *Theor. Comput. Sci.*, 221(1-2):211–250, 1999. doi:10.1016/S0304-3975(99)00033-X.

8 Zakaria Bouziane and Alain Finkel. Cyclic Petri Net Reachability Sets are Semi-Linear Effectively Constructible. In Faron Moller, editor, *Proceedings of the 2nd International Workshop on Verification of Infinite State Systems (INFINITY'97)*, volume 9 of *Electronic Notes in Theoretical Computer Science*, pages 15–24, Bologna, Italy, July 1997. Elsevier Science Publishers. URL: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PDF/BF-infinity97.pdf.

9 Marius Bozga, Radu Iosif, and Filip Konecný. Safety Problems are NP-complete for Flat Integer Programs with Octagonal Loops. *CoRR*, abs/1307.5321, 2013. arXiv:1307.5321.

10 Marius Bozga, Radu Iosif, Filip Konecný, and Tomás Vojnar. Tool Demonstration of the FLATA Counter Automata Toolset. In Andrei Voronkov, Laura Kovács, and Nikolaj Bjørner, editors, *Second International Workshop on Invariant Generation, WING 2009, York, UK, March 29, 2009 and Third International Workshop on Invariant Generation, WING 2010, Edinburgh, UK, July 21, 2010*, volume 1 of *EPiC Series in Computing*, page 75. EasyChair, 2010. URL: http://www.easychair.org/publications/paper/51875.

11 Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *J. ACM*, 30(2):323–342, 1983. doi:10.1145/322374.322380.

12 Nadia Busi, Roberto Gorrieri, Claudio Guidi, Roberto Lucchi, and Gianluigi Zavattaro. Choreography and Orchestration Conformance for System Design. In Paolo Ciancarini and Herbert Wiklicky, editors, *Coordination Models and Languages, 8th International Conference, COORDINATION 2006, Bologna, Italy, June 14-16, 2006, Proceedings*, volume 4038 of *Lecture Notes in Computer Science*, pages 63–81. Springer, 2006. doi:10.1007/11767954_5.

13 Gérard Cécé and Alain Finkel. Verification of Programs with Half-Duplex Communication. *Information and Computation*, 202(2):166–190, November 2005. doi:10.1016/j.ic.2005.05.006.

14 Normann Decker, Peter Habermehl, Martin Leucker, Arnaud Sangnier, and Daniel Thoma. Model-checking Counting Temporal Logics on Flat Structures. In *28th International Conference*

on Concurrency Theory, CONCUR 2017, LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

15 S. Demri, A. Finkel, V. Goranko, and G. van Drimmelen. Towards a Model-Checker for Counter Systems. In Susanne Graf and Wenhui Zhang, editors, *Automated Technology for Verification and Analysis*, pages 493–507, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

16 Stéphane Demri, Amit Dhar, and Arnaud Sangnier. Equivalence Between Model-Checking Flat Counter Systems and Presburger Arithmetic. *Theoretical Computer Science*, 2017. Special issue of RP'14, to appear.

17 Stéphane Demri, Amit Kumar Dhar, and Arnaud Sangnier. On the Complexity of Verifying Regular Properties on Flat Counter Systems. In Fedor V. Fomin, Rūsiņš Freivalds, Marta Kwiatkowska, and David Peleg, editors, *Proceedings of the 40th International Colloquium on Automata, Languages and Programming (ICALP'13) – Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 162–173, Riga, Latvia, July 2013. Springer. doi:10.1007/978-3-642-39212-2_17.

18 Stéphane Demri, Amit Kumar Dhar, and Arnaud Sangnier. Taming past LTL and flat counter systems. *Inf. Comput.*, 242:306–339, 2015. doi:10.1016/j.ic.2015.03.007.

19 Stéphane Demri, Alain Finkel, Valentin Goranko, and Govert van Drimmelen. Model-checking CTL* over Flat Presburger Counter Systems. *Journal of Applied Non-Classical Logics*, 20(4):313–344, 2010. doi:10.3166/jancl.20.313-344.

20 Frank Drewes and Jérôme Leroux. Structurally Cyclic Petri Nets. *Logical Methods in Computer Science*, 11(4), 2015. doi:10.2168/LMCS-11(4:15)2015.

21 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.

22 Alain Finkel. *Structuration des systèmes de transitions: applications au contrôle du parallélisme par files fifo, Thèse d'Etat.* PhD thesis, Université Paris-Sud, Orsay, 1986.

23 Alain Finkel and Jean Goubault-Larrecq. Forward Analysis for WSTS, Part II: Complete WSTS. *Logical Methods in Computer Science*, 8(3:28), September 2012. doi:10.2168/LMCS-8(3:28)2012.

24 Alain Finkel and Étienne Lozes. Synchronizability of Communicating Finite State Machines is not Decidable. In Ioannis Chatzigiannakis, Piotr Indyk, Anca Muscholl, and Fabian Kuhn, editors, *Proceedings of the 44th International Colloquium on Automata, Languages and Programming (ICALP'17)*, volume 80 of *Leibniz International Proceedings in Informatics*, pages 122:1–122:14, Warsaw, Poland, July 2017. Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2017.122.

25 Alain Finkel, S. Purushothaman Iyer, and Grégoire Sutre. Well-Abstracted Transition Systems: Application to FIFO Automata. *Information and Computation*, 181(1):1–31, February 2003. URL: http://www.lsv.ens-cachan.fr/Publis/PAPERS/PS/FPS-ICOMP.ps.

26 Blaise Genest, Dietrich Kuske, and Anca Muscholl. On Communicating Automata with Bounded Channels. *Fundam. Inform.*, 80(1-3):147–167, 2007. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi80-1-3-09.

27 Christoph Haase. *On the complexity of model checking counter automata.* PhD thesis, University of Oxford, UK, 2012.

28 Radu Iosif and Arnaud Sangnier. How Hard is It to Verify Flat Affine Counter Systems with the Finite Monoid Property? In Cyrille Artho, Axel Legay, and Doron Peled, editors, *Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings*, volume 9938 of *Lecture Notes in Computer Science*, pages 89–105, 2016. doi:10.1007/978-3-319-46520-3_6.

29 Thierry Jéron and Claude Jard. Testing for Unboundedness of FIFO Channels. *Theor. Comput. Sci.*, 113(1):93–117, 1993. doi:10.1016/0304-3975(93)90212-C.

30 Julien Lange and Nobuko Yoshida. Verifying Asynchronous Interactions via Communicating Session Automata. *CoRR*, abs/1901.09606, 2019. `arXiv:1901.09606`.

31 Christos H. Papadimitriou. On the Complexity of Integer Programming. *J. ACM*, 28(4):765–768, October 1981. `doi:10.1145/322276.322287`.

32 Gregoire Sutre. Personal communication, 2018.

33 Gregor von Bochmann. Communication protocols and error recovery procedures. *Operating Systems Review*, 9(3):45–50, 1975.

The Complexity of Subgame Perfect Equilibria in Quantitative Reachability Games

Thomas Brihaye
Université de Mons (UMONS), Belgium

Véronique Bruyère
Université de Mons (UMONS), Belgium

Aline Goeminne
Université de Mons (UMONS), Belgium
Université libre de Bruxelles (ULB), Belgium

Jean-François Raskin
Université libre de Bruxelles (ULB), Belgium

Marie van den Bogaard
Université libre de Bruxelles (ULB), Belgium

Abstract

We study multiplayer quantitative reachability games played on a finite directed graph, where the objective of each player is to reach his target set of vertices as quickly as possible. Instead of the well-known notion of Nash equilibrium (NE), we focus on the notion of subgame perfect equilibrium (SPE), a refinement of NE well-suited in the framework of games played on graphs. It is known that there always exists an SPE in quantitative reachability games and that the constrained existence problem is decidable. We here prove that this problem is PSPACE-complete. To obtain this result, we propose a new algorithm that iteratively builds a set of constraints characterizing the set of SPE outcomes in quantitative reachability games. This set of constraints is obtained by iterating an operator that reinforces the constraints up to obtaining a fixpoint. With this fixpoint, the set of SPE outcomes can be represented by a finite graph of size at most exponential. A careful inspection of the computation allows us to establish PSPACE membership.

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 multiplayer non-zero-sum games played on graphs, quantitative reachability objectives, subgame perfect equilibria, constrained existence problem

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.13

Related Version A full version of the paper is available at `https://arxiv.org/abs/1905.00784`.

Funding Work 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 & FWO), and the COST Action 16228 *GAMENET* (European Cooperation in Science and Technology).

1 Introduction

While two-player zero-sum games played on graphs are the most studied model to formalize and solve the reactive synthesis problem [26], recent work has considered non-zero-sum extensions of this mathematical framework, see e.g. [15, 19, 7, 22, 6, 5, 17, 4, 1], see also the surveys [21, 3, 13]. In the zero-sum game approach, the system and the environment are considered as *monolithic* and fully *adversarial* entities. Unfortunately, both assumptions may turn to be too strong. First, the reactive system may be composed of several components

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 13; pp. 13:1–13:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

that execute concurrently and have their own purpose. So, it is natural to model such systems with *multiplayer* games with each player having his own objective. Second, the environment usually has its own objective too, and this objective is usually not the negation of the objective of the reactive system as postulated in the zero-sum case. Therefore, there are instances of the reactive synthesis problem for which no solution exists in the zero-sum setting, i.e. no winning strategy for the system against a completely antagonistic environment, while there exists a strategy for the system which enforces the desired properties against all *rational* behaviors of the environment pursuing its own objective.

While the central solution concept in zero-sum games is the notion of *winning strategy*, it is well known that this concept is not sufficient to reason about non-zero-sum games. In non-zero-sum games, notions of equilibria are used to reason about the rational behavior of players. The celebrated notion of *Nash equilibrium* (NE) [24] is one of the most studied. A strategy profile is an NE if no player has an incentive to deviate, i.e. change his strategy and obtain a better reward, when this player knows that the other players will be playing their respective strategies in the profile. A well-known weakness of NE in sequential games, which include infinite duration games played on graphs, is that they are subject to *non-credible threats*: decisions in subgames that are irrational and used to threaten the other players and oblige them to follow a given behavior. To avoid this problem, the concept of *subgame perfect equilibria* (SPE) has been proposed, see e.g. [25]. SPEs are NEs with the additional property that they are also NEs in all subgames of the original game. While it is now quite well understood how to handle NEs algorithmically in games played on graphs [28, 29, 12, 17], this is not the case for SPEs.

Contributions. In this paper, we provide an algorithm to decide in *polynomial space* the constrained existence problem for SPEs in *quantitative reachability games*. A quantitative reachability game is played by n players on a finite graph in which each player has his own reachability objective. The objective of each player is to reach his target set of vertices as quickly as possible. In a series of papers devoted to quantitative reachability games, it has been shown that SPEs always exist [8], and that the set of SPE outcomes is a regular language which is effectively constructible [11]. As a consequence of the latter result, the constrained existence problem for SPEs is decidable.

Unfortunately, the proof that establishes the regularity of the set of possible SPE outcomes in [11] exploits a *well-quasi order* for proving termination that cannot be used to obtain a good upper bound on the complexity for the algorithm. Here, we propose a new algorithm and we show that this set of outcomes can be represented using an automaton of size at most exponential. It follows that the constrained existence problem for SPEs is in PSPACE. We also provide a matching lower-bound showing that this problem is PSPACE-complete.

Our new algorithm iteratively builds a set of constraints that exactly characterize the set of SPEs in quantitative reachability games. This set of constraints is obtained by iterating an operator that reinforces the constraints up to obtaining a fixpoint. A careful inspection of the computation allows us to establish PSPACE membership.

Related work. Algorithms to reason on NEs in graph games are studied in [28] for ω-regular objectives and in [29, 12] for quantitative objectives. Algorithms to reason on SPEs are given in [27] for ω-regular objectives. Quantitative reachability objectives are not ω-regular objectives. Reasoning about NEs and SPEs for ω-regular specifications can also be done using strategy logics [16, 23]. Other notions of rationality used for reactive synthesis have been studied in the literature: rational synthesis in cooperative [19] and adversarial [22] setting, and their algorithmic complexity in [17]. Extensions with imperfect information have been investigated in [18]. Synthesis rules based on the notion of admissible strategies have been studied in [2, 7, 6, 5, 4, 1]. Weak SPEs have been studied in [11, 14, 9] and shown to be equivalent to SPEs for quantitative reachability objectives. Fixpoint techniques are used in [20, 11, 14] to establish the existence of (weak) SPEs in some classes of games, however they cannot be used in our context to get complexity results.

2 Preliminaries

In this section, we recall the notions of quantitative reachability game and subgame perfect equilibrium. We also state the problem studied in this paper and our main result.

Quantitative reachability games. An *arena* is a tuple $G = (\Pi, V, (V_i)_{i\in\Pi}, E)$ where $\Pi = \{1, 2, \dots, n\}$ is a finite set of n players, V is a finite set of vertices with $|V| \geq 2$, $(V_i)_{i\in\Pi}$ is a partition of V between the players, and $E \subseteq V \times V$ is a set of edges such that for all $v \in V$ there exists $v' \in V$ such that $(v, v') \in E$. Without loss of generality, we suppose that $|\Pi| \leq |V|$. We denote by $\text{Succ}(v) = \{v' \mid (v, v') \in E\}$ the set of *successors* of v, for $v \in V$, and by Succ^* the transitive closure of Succ.

A *play* in G is an infinite sequence of vertices $\rho = \rho_0\rho_1 \dots$ such that for all $k \in \mathbb{N}$, $(\rho_k, \rho_{k+1}) \in E$. A *history* is a finite sequence $h = h_0h_1 \dots h_k$ with $k \in \mathbb{N}$ defined similarly. The *length* $|h|$ of h is the number k of its edges. We denote the set of plays by Plays and the set of histories by Hist (when it is necessary, we use notation Plays_G and Hist_G to recall the underlying arena G). Moreover, the set Hist_i is the set of histories such that their last vertex v is a vertex of player i, i.e. $v \in V_i$.

Given a play $\rho \in \text{Plays}$ and $k \in \mathbb{N}$, the prefix $\rho_0\rho_1 \dots \rho_k$ of ρ is denoted by $\rho_{\leq k}$ and its suffix $\rho_k\rho_{k+1} \dots$ by $\rho_{\geq k}$. A play ρ is called a *lasso* if it is of the form $\rho = h\ell^\omega$ with $h\ell \in \text{Hist}$. Notice that ℓ is not necessary a simple cycle. The *length* of a lasso $h\ell^\omega$ is the length of $h\ell$.

A *quantitative game* $\mathcal{G} = (G, (\text{Cost}_i)_{i\in\Pi})$ is an arena equipped with a cost function profile $\text{Cost} = (\text{Cost}_i)_{i\in\Pi}$ such that each function $\text{Cost}_i : \text{Plays} \to \mathbb{R} \cup \{+\infty\}$ assigns a cost to each play. In a quantitative game $\mathcal{G}$, an initial vertex $v_0 \in V$ is often fixed, and we call $(\mathcal{G}, v_0)$ an *initialized game*. A play (resp. a history) of $(\mathcal{G}, v_0)$ is then a play (resp. a history) of $\mathcal{G}$ starting in v_0. The set of such plays (resp. histories) is denoted by $\text{Plays}(v_0)$ (resp. $\text{Hist}(v_0)$). We also use notation $\text{Hist}_i(v_0)$ when these histories end in a vertex $v \in V_i$.

In this article we are interested in *quantitative reachability games* such that each player has a target set of vertices that he wants to reach. The cost to pay is equal to the number of edges to reach the target set, and each player aims at minimizing his cost.

▶ **Definition 1** (Quantitative reachability game). *A* quantitative reachability game *(or simply a* reachability game*) is a tuple $\mathcal{G} = (G, (F_i)_{i\in\Pi}, (\text{Cost}_i)_{i\in\Pi})$ such that (i) G is an arena, (ii) for each $i \in \Pi$, $F_i \subseteq V$ is the target set of player i, and (iii) for each $i \in \Pi$ and each $\rho = \rho_0\rho_1 \dots \in \text{Plays}$, $\text{Cost}_i(\rho)$ is equal to the least index k such that $\rho_k \in F_i$, and to $+\infty$ if no such index exists.*

Given a quantitative game $\mathcal{G}$, a *strategy* for player i is a function $\sigma_i : \text{Hist}_i \to V$. It assigns to each history hv, with $v \in V_i$, a vertex v' such that $(v, v') \in E$. In an initialized game $(\mathcal{G}, v_0)$, σ_i needs only to be defined for histories starting in v_0. A play $\rho = \rho_0\rho_1 \dots$ is *consistent* with σ_i if for all $\rho_k \in V_i$, $\sigma_i(\rho_0 \dots \rho_k) = \rho_{k+1}$. A strategy σ_i is *positional* if it only depends on the last vertex of the history, *i.e.*, $\sigma_i(hv) = \sigma_i(v)$ for all $hv \in \text{Hist}_i$.

A *strategy profile* is a tuple $\sigma = (\sigma_i)_{i\in\Pi}$ of strategies, one for each player. Given an initialized game $(\mathcal{G}, v_0)$ and a strategy profile σ, there exists a unique play from v_0 consistent with each strategy σ_i. We call this play the *outcome* of σ and denote it by $\langle\sigma\rangle_{v_0}$. Let $c = (c_i)_{i\in\Pi} \in (\mathbb{N} \cup \{+\infty\})^{|\Pi|}$, we say that σ is a strategy profile *with cost* c or that $\langle\sigma\rangle_{v_0}$ *has cost* c if $c_i = \mathrm{Cost}_i(\langle\sigma\rangle_{v_0})$ for all $i \in \Pi$.

Solution concepts and constraint problem. In the multiplayer game setting, the solution concepts usually studied are *equilibria* (see [21]). We here recall the concepts of Nash equilibrium and subgame perfect equilibrium.

Let $\sigma = (\sigma_i)_{i\in\Pi}$ be a strategy profile in a game $(\mathcal{G}, v_0)$. When we highlight the role of player i, we denote σ by (σ_i, σ_{-i}) where σ_{-i} is the profile $(\sigma_j)_{j\in\Pi\setminus\{i\}}$. A strategy $\sigma'_i \neq \sigma_i$ is a *deviating* strategy of player i, and it is a *profitable deviation* for him if $\mathrm{Cost}_i(\langle\sigma\rangle_{v_0}) > \mathrm{Cost}_i(\langle\sigma'_i, \sigma_{-i}\rangle_{v_0})$.

The notion of Nash equilibrium is classical: a strategy profile σ in an initialized game $(\mathcal{G}, v_0)$ is a *Nash equilibrium* (NE) if no player has an incentive to deviate unilaterally from his strategy, i.e. no player has a profitable deviation. Formally, σ is an NE if for each $i \in \Pi$ and each deviating strategy σ'_i of player i, we have $\mathrm{Cost}_i(\langle\sigma\rangle_{v_0}) \leq \mathrm{Cost}_i(\langle\sigma'_i, \sigma_{-i}\rangle_{v_0})$.

When considering games played on graphs, a useful refinement of NE is the concept of *subgame perfect equilibrium* (SPE) which is a strategy profile being an NE in each subgame. It is well-known that contrarily to NEs, SPEs avoid non-credible threats [21]. Formally, given a quantitative game $\mathcal{G} = (G, \mathrm{Cost})$, an initial vertex v_0, and a history $hv \in \mathrm{Hist}(v_0)$, the initialized game $(\mathcal{G}_{\restriction h}, v)$ is called a *subgame* of $(\mathcal{G}, v_0)$ such that $\mathcal{G}_{\restriction h} = (G, \mathrm{Cost}_{\restriction h})$ and $\mathrm{Cost}_{i\restriction h}(\rho) = \mathrm{Cost}_i(h\rho)$ for all $i \in \Pi$ and $\rho \in V^\omega$. Notice that $(\mathcal{G}, v_0)$ is subgame of itself. Moreover if σ_i is a strategy for player i in $(\mathcal{G}, v_0)$, then $\sigma_{i\restriction h}$ denotes the strategy in $(\mathcal{G}_{\restriction h}, v)$ such that for all histories $h' \in \mathrm{Hist}_i(v)$, $\sigma_{i\restriction h}(h') = \sigma_i(hh')$. Similarly, from a strategy profile σ in $(\mathcal{G}, v_0)$, we derive the strategy profile $\sigma_{\restriction h}$ in $(\mathcal{G}_{\restriction h}, v)$.

▶ **Definition 2** (Subgame perfect equilibrium). *A strategy profile σ is a* subgame perfect equilibrium *in an initialized game $(\mathcal{G}, v_0)$ if for all $hv \in \mathrm{Hist}(v_0)$, $\sigma_{\restriction h}$ is an NE in $(\mathcal{G}_{\restriction h}, v)$.*

It is proved in [8] that there always exists an SPE in reachability games. In this paper, we are interested in solving the following *constraint problem*.

▶ **Definition 3** (Constraint problem). *Given $(\mathcal{G}, v_0)$ an initialized reachability game and two threshold vectors $x, y \in (\mathbb{N} \cup \{+\infty\})^{|\Pi|}$, the constraint problem is to decide whether there exists an SPE in $(\mathcal{G}, v_0)$ with cost c such that $x \leq c \leq y$, that is, $x_i \leq c_i \leq y_i$ for all $i \in \Pi$.*

Our main result is the following one. The paper is devoted to its proof.

▶ **Theorem 4.** *The constraint problem for initialized reachability games is PSPACE-complete.*

▶ **Example 5.** A reachability game $\mathcal{G}$ with two players is depicted in Figure 1. The circle (resp. square) vertices are owned by player 1 (resp. player 2). The target sets of both players are respectively $F_1 = \{v_2\}$ (grey vertex) and $F_2 = \{v_2, v_5\}$ (double circled vertices).

The positional strategy profile $\sigma = (\sigma_1, \sigma_2)$ is depicted by double arrows, its outcome in $(\mathcal{G}, v_0)$ is equal to $\langle\sigma\rangle_{v_0} = (v_0v_1v_6v_7v_2)^\omega$ with cost $(4, 4)$. Let us explain that σ is an NE. Player 1 reaches his target set as soon as possible and has thus no incentive to deviate. Player 2 has no profitable deviation that allows him to reach v_5. For instance if he uses a deviating positional strategy σ'_2 such that $\sigma'_2(v_0) = v_4$, then the outcome of (σ_1, σ'_2) is equal to $(v_0v_4)^\omega$ with cost $(+\infty, +\infty)$ which is not profitable for player 2. One can verify that the strategy profile σ is also an SPE. For instance in the subgame $(\mathcal{G}_{\restriction h}, v_5)$ with $h = v_0v_4$, we have $\rho = \langle\sigma_{\restriction h}\rangle_{v_5} = v_5v_4(v_0v_1v_6v_7v_2)^\omega$ such that $\mathrm{Cost}_{\restriction h}(\rho) = \mathrm{Cost}(h\rho) = (8, 2)$. In this subgame, with ρ, both players reach their target set as soon as possible. ◀

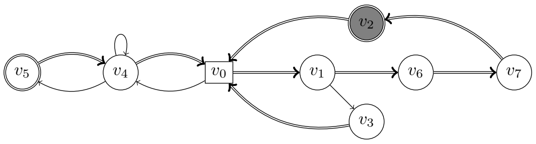


Figure 1 A quantitative reachability game: player 1 (resp. player 2) owns circle (resp. square) vertices and $F_1 = \{v_2\}$ and $F_2 = \{v_2, v_5\}$.

Extended game. We here present the *extended game* of a reachability game, such that the vertices are enriched with the set of players that have already visited their target sets along a history (see e.g. [9]). Working with this extended game is essential to prove our main result.

▶ **Definition 6** (Extended game). *Let $\mathcal{G} = (G, (F_i)_{i\in\Pi}, (\mathrm{Cost}_i)_{i\in\Pi})$ be a reachability game with an arena $G = (\Pi, V, (V_i)_{i\in\Pi}, E)$, and let v_0 be an initial vertex. The* extended game *of $\mathcal{G}$ is equal to $\mathcal{X} = (X, (F_i^X)_{i\in\Pi}, (\mathrm{Cost}_i)_{i\in\Pi})$ with the arena $X = (\Pi, V^X, (V_i^X)_{i\in\Pi}, E^X)$, such that:*

- $V^X = V \times 2^\Pi$
- $((v, I), (v', I')) \in E^X$ *if and only if* $(v, v') \in E$ *and* $I' = I \cup \{i \in \Pi \mid v' \in F_i\}$
- $(v, I) \in V_i^X$ *if and only if* $v \in V_i$
- $(v, I) \in F_i^X$ *if and only if* $i \in I$
- *for each* $\rho \in \mathrm{Plays}_X$, $\mathrm{Cost}_i(\rho)$ *is equal to the least index k such that $\rho_k \in F_i^X$, and to $+\infty$ if no such index exists.*

The initialized extended game $(\mathcal{X}, x_0)$ associated with the initialized game $(\mathcal{G}, v_0)$ is such that $x_0 = (v_0, I_0)$ with $I_0 = \{i \in \Pi \mid v_0 \in F_i\}$.

Notice the way each target set F_i^X is defined: if $v \in F_i$, then $(v, I) \in F_i^X$ but also $(v', I') \in F_i^X$ for all $(v', I') \in \mathrm{Succ}^*(v, I)$. The extended game of the reachability game of Figure 1 is depicted in Figure 2 (until Section 3 the reader should not consider the labeling indicated close to the vertices). We will come back to this example at the end of this section.

Let us state some properties of the extended game. First, notice that for each $\rho = (v_0, I_0)(v_1, I_1)\ldots \in \mathrm{Plays}_X(x_0)$, we have the next property called *I-monotonicity*:

$$I_k \subseteq I_{k+1} \qquad \text{for all } k \in \mathbb{N}. \tag{1}$$

Second, given an initialized game $(\mathcal{G}, v_0)$ and its extended game $(\mathcal{X}, x_0)$, there is a one-to-one correspondence between plays in $\mathrm{Plays}_G(v_0)$ and plays in $\mathrm{Plays}_X(x_0)$:

- from $\rho = \rho_0\rho_1\ldots \in \mathrm{Plays}_G(v_0)$, we derive $\rho^X = (\rho_0, I_0)(\rho_1, I_1)\ldots \in \mathrm{Plays}_X(x_0)$ such that I_k is the set of players i that have seen their target set F_i along $\rho_{\leq k}$;
- from $\rho = (v_0, I_0)(v_1, I_1)\ldots \in \mathrm{Plays}_X(x_0)$, we derive $\rho^G = v_0v_1\ldots \in \mathrm{Plays}_G(v_0)$ such that the second components I_k, $k \in \mathbb{N}$, are omitted.

Third, given $\rho \in \mathrm{Plays}_G(v_0)$, we have that $\mathrm{Cost}(\rho^X) = \mathrm{Cost}(\rho)$, and conversely given $\rho \in \mathrm{Plays}_X(x_0)$, we have that $\mathrm{Cost}(\rho^G) = \mathrm{Cost}(\rho)$. It follows that outcomes of SPE can be equivalently studied in $(\mathcal{G}, v_0)$ and in $(\mathcal{X}, x_0)$, as stated in the next lemma.

▶ **Lemma 7.** *If ρ is the outcome of an SPE in $(\mathcal{G}, v_0)$, then ρ^X is the outcome of an SPE in $(\mathcal{X}, x_0)$ with the same cost. Conversely, if ρ is the outcome of an SPE in $(\mathcal{X}, x_0)$, then ρ^G is the outcome of an SPE in $(\mathcal{G}, v_0)$ with the same cost.*

By construction, the arena X of the initialized extended game is divided into different regions according to the players who have already visited their target set. Let us provide some useful notions with respect to this decomposition. We will often use them in the following sections. Let $\mathcal{I} = \{I \subseteq \Pi \mid \text{there exists } v \in V \text{ such that } (v, I) \in \mathrm{Succ}^*(x_0)\}$ be the

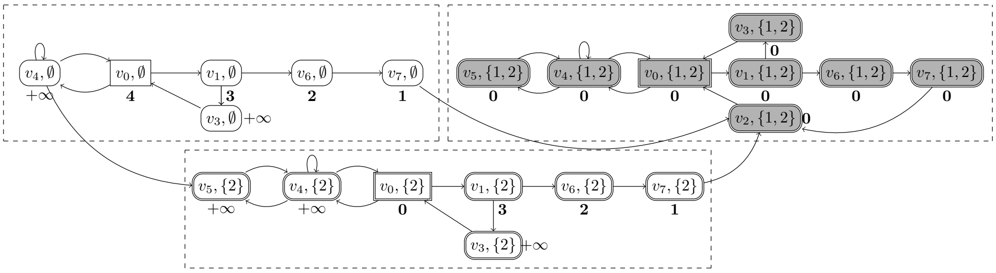


Figure 2 The extended game $(\mathcal{X}, x_0)$ for the initialized game (G, v_0) of Figure 1. The values of a labeling function λ are indicated close to each vertex.

set of sets I accessible from the initial state x_0, and let $N = |\mathcal{I}|$ be its size. For $I, I' \in \mathcal{I}$ with $I \neq I'$, if there exists $((v, I), (v', I')) \in E^X$, we say that I' is a *successor* of I and we write $I' \in \mathrm{Succ}(I)$. Given $I \in \mathcal{I}$, $X^I = (V^I, E^I)$ refers to the sub-arena of X restricted to the vertices $\{(v, I) \in V^X \mid v \in V\}$. We say that X^I is the *region*[1] associated with I. Such a region X^I is called a *bottom region* whenever $\mathrm{Succ}(I) = \emptyset$.

There exists a partial order on $\mathcal{I}$ such that $I < I'$ if and only if $I' \in \mathrm{Succ}^+(I)$.

We fix an arbitrary *total order* on $\mathcal{I}$ that extends this partial order $<$ as follows:

$$J_1 < J_2 < \ldots < J_N. \tag{2}$$

(with X^{J_N} a bottom region).[2] With respect to this total order, given $n \in \{1, \ldots, N\}$, we denote by $X^{\geq J_n} = (V^{\geq J_n}, E^{\geq J_n})$ the sub-arena of X restricted to the vertices $\{(v, I) \in V^X \mid I \geq J_n\}$. This total order together with the I-monotonicity leads to the following lemma.

▶ **Lemma 8** (Region decomposition and section). *Let π be a (finite or infinite) path in $\mathcal{X}$. Then there exists a* region decomposition *of π as $\pi[\ell]\pi[\ell+1]\ldots\pi[m]$ with $1 \leq \ell \leq m \leq N$, such that for each n, $\ell \leq n \leq m$:*

- *$\pi[n]$ is a (possibly empty) path in $\mathcal{X}$,*
- *every vertex of $\pi[n]$ is of the form (v, J_n) for some $v \in V$.*

Each path $\pi[n]$ is called a section*. The last section $\pi[m]$ is infinite if and only if π is infinite.*

▶ **Example 9.** Let us come back to the game $(\mathcal{G}, v_0)$ of Figure 1. Its extended game $(\mathcal{X}, x_0)$ is depicted in Figure 2 (only the part reachable from the initial vertex $x_0 = (v_0, \emptyset)$ is depicted).

The extended game is divided into three different regions: one region associated to $I = \emptyset$ that contains x_0, a second region associated to $I = \{2\}$, and a third bottom region associated to $I = \Pi$. Hence the set $\mathcal{I} = \{\emptyset, \{2\}, \Pi\}$ is totally ordered as $J_1 = \emptyset < J_2 = \{2\} < J_3 = \Pi$. For all the vertices (v, I) of the region associated with $I = \{2\}$, we have $(v, I) \notin F_1^X$ and $(v, I) \in F_2^X$, and for those of the region associated with $I = \Pi$, we have $(v, I) \in F_1^X \cap F_2^X$.

From the SPE σ given in Example 5 with outcome $\rho = (v_0 v_1 v_6 v_7 v_2)^\omega \in \mathrm{Plays}_G(v_0)$ and cost $(4, 4)$, we derive the SPE outcome $\rho^X \in \mathrm{Plays}_X(x_0)$ with the same cost and equal to

$$(v_0, \emptyset)(v_1, \emptyset)(v_6, \emptyset)(v_7, \emptyset)((v_2, \Pi)(v_0, \Pi)(v_1, \Pi)(v_6, \Pi)(v_7, \Pi))^\omega \tag{3}$$

[1] In the sequel, we indifferently call region either X^I, or V^I, or I.

[2] We use notation J_n, $n \in \{1, \ldots, N\}$, to avoid any confusion with the sets I_k appearing in a play $\rho = (v_0, I_0)(v_1, I_1)\ldots$.

3 Characterization

In this section, given an initialized reachability game $(\mathcal{G}, v_0)$, we characterize the set of plays that are outcomes of SPEs, and we provide an algorithm to construct this set. For this characterization, by Lemma 7, we can work on the extended game $(\mathcal{X}, x_0)$ instead of $(\mathcal{G}, v_0)$.

All along this section, when we refer to a vertex of V^X, we use notation v (instead of (u, I)) and notation $I(v)$ means the second component I of this vertex.

Our algorithm iteratively builds a set of *constraints* imposed by a *labeling function* $\lambda : V^X \to \mathbb{N} \cup \{+\infty\}$ such that the plays of the extended game satisfying those constraints are exactly the SPE outcomes. Let us provide a formal definition of such a function λ with the constraints that it imposes on plays.

▶ **Definition 10** (λ-consistent play). *Let $\mathcal{X}$ be the extended game of a reachability game $\mathcal{G}$, and $\lambda : V^X \to \mathbb{N} \cup \{+\infty\}$ be a labeling function. Given $v \in V^X$, for all plays $\rho \in \mathrm{Plays}_X(v)$, we say that $\rho = \rho_0\rho_1 \dots$ is* λ-consistent *if for all $n \in \mathbb{N}$ and $i \in \Pi$ such that $\rho_n \in V_i$:*

$$\mathrm{Cost}_i(\rho_{\geq n}) \leq \lambda(\rho_n). \tag{4}$$

We denote by $\Lambda(v)$ the set of plays $\rho \in \mathrm{Plays}_X(v)$ that are λ-consistent.

Thus, a play ρ is λ-consistent if for all its suffixes $\rho_{\geq n}$, if player i owns ρ_n then the number of edges to reach his target set along $\rho_{\geq n}$ is bounded by $\lambda(\rho_n)$. Before going into the details of our algorithm, let us intuitively explain on an example how a well-chosen labeling function characterizes the set of SPE outcomes.

▶ **Example 11.** We consider the extended game $(\mathcal{X}, x_0)$ of Figure 2, and a labeling function λ whose values are indicated under or next to each vertex. If $v \in V_i^X$ is labeled by $\lambda(v) = c$, then if $c \in \mathbb{N}_0$, this means that player i will only accept outcomes in $(\mathcal{X}, v)$ that reach his target set within c steps, otherwise he would have a profitable deviation. If $\lambda(v) = 0$, this means that player i has already reached his target set, and if $\lambda(v) = +\infty$, player i has no profitable deviation whatever outcome is proposed to him.

In Example 9 was given the SPE outcome equal to ρ^X (3) and with cost $(4, 4)$. We have $\lambda(v_0, \emptyset) = 4$ and player 2 reaches his target set from $(v_0, \emptyset)$ within exactly 4 steps. The constraints imposed by λ on the other vertices of ρ are respected too. On the other hand, one can prove that $\rho' = ((v_0, \emptyset)(v_4, \emptyset))^\omega$ is the outcome of no SPE. It is not λ-consistent since player 2 does not reach his target set, and so in particular not within 4 steps. ◀

Our algorithm roughly works as follows: the labeling function λ that characterizes the set of SPE outcomes is obtained *(i)* from an initial labeling function that imposes no constraints, *(ii)* by iterating an operator that reinforces the constraints step after step, *(iii)* up to obtaining a fixpoint which is the required function λ. Thus, if λ^k is the labeling function computed at step k and $\Lambda^k(v)$, $v \in V^X$, the related sets of λ^k-consistent plays, initially we have $\Lambda^0(v) = \mathrm{Plays}_X(v)$, and step by step, the constraints imposed by λ^k become stronger and the sets $\Lambda^k(v)$ become smaller, until a fixpoint is reached.

Initially, we want a labeling function λ^0 that imposes no constraint in a way to have $\Lambda^0(v) = \mathrm{Plays}_X(v)$. We define $\lambda^0(v) = +\infty$ except when $i \in I(v)$ and $v \in V_i^X$ where $\lambda^0(v) = 0$. Indeed by Definition 6, we have $v \in F_i^X$ if and only if $i \in I(v)$. Hence, given $\rho = \rho_0\rho_1 \dots$, once $\rho_k \in F_i^X$ for some $k \in \mathbb{N}$ then $\rho_n \in F_i^X$ for all $n \geq k$. It follows that for all $n \geq k$, $\mathrm{Cost}_i(\rho_{\geq n}) = 0$ and the inequality (4) is trivially true. (See also Example 11.)

▶ **Definition 12** (Initial labeling). *For all $v \in V^X$, let $i \in \Pi$ be such that $v \in V_i^X$,*

$$\lambda^0(v) = 0 \text{ if } i \in I(v) \quad \text{and} \quad \lambda^0(v) = +\infty \text{ otherwise.}$$

Let us now explain how our algorithm computes the labeling functions λ^k, $k \geq 1$, and the related sets $\Lambda^k(v)$, $v \in V^X$. It works in a *bottom-up* manner, according to the total order $J_1 < J_2 < \ldots < J_N$ of $\mathcal{I}$ given in (2). It first iteratively updates the labeling function for all vertices v of the arena X^{J_N}. At some point, the values of λ^k do not change anymore in X^{J_N} and $(\lambda^k)_{k\in\mathbb{N}}$ reaches locally (on X^{J_N}) a fixpoint. Then it treats the arena $X^{\geq J_{N-1}}$ and in the same way, it updates locally the values of λ^k in $X^{\geq J_{N-1}}$ until reaching a (local) fixpoint. It then repeats this procedure in $X^{\geq J_{N-2}}$, ..., $X^{\geq J_1} = X$.

Hence suppose we currently treat the arena $X^{\geq J_n}$ and we want to compute λ^{k+1} from λ^k. We define the updated function λ^{k+1} as follows (with the convention that $1 + (+\infty) = +\infty$).

▶ **Definition 13** (Labeling update). *Let $k \geq 0$ and suppose that we treat the arena $X^{\geq J_n}$, with $n \in \{1, \ldots, N\}$. For all $v \in V^X$,*

- *if $v \in V^{\geq J_n}$, let $i \in \Pi$ be such that $v \in V_i^X$, then*

$$\lambda^{k+1}(v) = 0 \textit{ if } i \in I(v) \quad \textit{and} \quad \lambda^{k+1}(v) = 1 + \min_{(v,v')\in E^X} \sup\{\mathrm{Cost}_i(\rho) \mid \rho \in \Lambda^k(v')\} \textit{ otherwise.}$$

- *if $v \notin V^{\geq J_n}$, then $\lambda^{k+1}(v) = \lambda^k(v)$.*

▶ Remark 14. If the sup in Definition 13 is equal to $+\infty$ then there exists $\rho \in \Lambda^k(v')$ such that $\mathrm{Cost}_i(\rho) = +\infty$. Thus the sup can be replaced by a max which belongs to $\mathbb{N} \cup \{+\infty\}$.

Let us provide some explanations. As this update concerns the arena $X^{\geq J_n}$, we keep $\lambda^{k+1} = \lambda^k$ outside of this arena. Suppose now that v belongs to the arena $X^{\geq J_n}$ and $v \in V_i^X$. We define $\lambda^{k+1}(v) = 0$ whenever $i \in I(v)$ (as already explained for the definition of λ^0). When it is updated, the value $\lambda^{k+1}(v)$ represents what is the best cost that player i can ensure for himself with a "one-shot" choice by only taking into account plays of $\Lambda^k(v')$ with $v' \in \mathrm{Succ}(v)$. Notice that it makes sense to run the algorithm in a bottom-up fashion according to the total ordering $J_1 < \ldots < J_N$ since given a play $\rho = \rho_0\rho_1\ldots$, if ρ_0 is a vertex of $V^{\geq J_n}$, then for all $k \in \mathbb{N}$, ρ_k is a vertex of $V^{\geq J_n}$ (by I-monotonicity). Moreover running the algorithm in this way is essential to prove that the constraint problem is in PSPACE.

We can now provide our algorithm that computes the sequence $(\lambda^k(v))_{k\in\mathbb{N}}$. From the last computed λ^k, we derive the sets $\Lambda^k(v)$, $v \in V^X$, that we need for the characterization of outcomes of SPEs (see Theorem 17 below). Such a characterization already appears in [11] however with a different algorithm that cannot be used to obtain good complexity upper bounds for the constraint problem as done in this paper. Nevertheless, the proof of our characterization and that of [11] are similar.

Algorithm 1: Fixpoint.

```
$k \leftarrow 0$; $n \leftarrow N$; compute $\lambda^0$ (see Definition 12)
while $n \neq 0$ do
    repeat
        $k \leftarrow k + 1$; compute $\lambda^k$ from $\lambda^{k-1}$ with respect to $X^{\geq J_n}$ (see Definition 13)
    until $\lambda^k = \lambda^{k-1}$
    $n \leftarrow n - 1$
end
return $\lambda^k$.
```

As already announced, the sequence $(\lambda^k)_{k\in\mathbb{N}}$ computed by this algorithm reaches a fixpoint – locally on each arena $X^{\geq J_n}$ and globally on X – in the following meaning:

▶ **Proposition 15.** *There exists a sequence* $0 = k^*_N < k^*_{N-1} < \ldots < k^*_1 = k^*$ *such that*

- ***Local fixpoint:*** *for all* $J_n \in \mathcal{I}$, *all* $m \in \mathbb{N}$ *and all* $v \in V^{\geq J_n}$: $\lambda^{k^*_n+m}(v) = \lambda^{k^*_n}(v)$.
- ***Global fixpoint:*** *with* $k^* = k^*_1$, *for all* $m \in \mathbb{N}$ *and all* $v \in V^X$: $\lambda^{k^*+m}(v) = \lambda^{k^*}(v)$.

The global fixpoint λ^{k^*} *is also simply denoted by* λ^*, *and the set* Λ^{k^*} *is denoted by* Λ^*.

The fixpoints follow from the existence of a well-quasi-ordering on the non increasing sequences $(\lambda^k(v))_{k\in\mathbb{N}}$. Proposition 15 indicates that Algorithm 1 terminates. Indeed for each $J_n \in \mathcal{I}$, taking the *least index* k^*_n such that $\lambda^{k^*_n+1}(v) = \lambda^{k^*_n}(v)$ for every $v \in X^{\geq J_n}$ shows that the repeat loop is broken and the variable n decremented by 1. The value $n = 0$ is eventually reached and the algorithm stops with the global fixpoint λ^*. Notice that the first local fixpoint is reached with $k^*_N = 0$ as X^{J_N} is a bottom region.

Proposition 15 also shows that when a local fixpoint is reached in the arena $X^{\geq J_{n+1}}$ and the algorithm updates the labeling function λ^k in the arena $X^{\geq J_n}$, the values of $\lambda^k(v)$ do not change anymore for any $v \in V^{\geq J_{n+1}}$ but can still be modified for some $v \in V^{J_n}$. Recall also that outside of $X^{\geq J_n}$, the values of $\lambda^k(v)$ are still equal to the initial values $\lambda^0(v)$. These properties will be useful when we will prove that the constraint problem for reachability games is in PSPACE. They are summarized in the next lemma.

▶ **Lemma 16.** *Let* k *be a step of Algorithm 1, let* J_n *with* $n \in \{1, \ldots, N\}$. *For all* $v \in V^{J_n}$:

- *if* $k \leq k^*_{n+1}$, *then* $\lambda^{k+1}(v) = \lambda^k(v) = \lambda^0(v)$,
- *if* $k^*_n \leq k$, *then* $\lambda^{k+1}(v) = \lambda^k(v) = \lambda^{k^*_n}(v)$,

Hence the values of $\lambda^k(v)$ *and* $\lambda^{k+1}(v)$ *may be different only when* $k^*_{n+1} < k < k^*_n$.

The following theorem states how we characterize outcomes of SPEs in $(\mathcal{X}, x_0)$. It also provides a characterization of the outcomes of SPEs in $(\mathcal{G}, v_0)$ by Lemma 7.

▶ **Theorem 17** (Characterization). *Let* $(\mathcal{G}, v_0)$ *be an initialized quantitative game and* $(\mathcal{X}, x_0)$ *be its extended game. Let* ρ *be a play in* $\mathrm{Plays}_X(x_0)$. *Then* ρ *is the outcome of an SPE in* $(\mathcal{X}, x_0)$ *if and only if for all* $v \in \mathrm{Succ}^*(x_0)$, $\Lambda^*(v) \neq \emptyset$ *and* $\rho \in \Lambda^*(x_0)$.

▶ **Example 18.** Let us come back to the running example of Figure 2. The different steps of Algorithm 1 are given in Table 1. The columns indicate the vertices according to their region, respectively Π, $\{2\}$, and $\emptyset$. Notice that for the region Π, we only write one column v as for all vertices (v, Π) the value of λ is equal to 0 all along the algorithm.

Recall that $J_1 = \emptyset < J_2 = \{2\} < J_3 = \Pi = \{1, 2\}$. The algorithm begins with the arena X^{J_3}. A fixpoint ($\lambda^1 = \lambda^0$) is immediately reached because all vertices belong to the target set of both players in X^{J_3}. Thus the first local fixpoint is reached with $k^*_3 = 0$.

The algorithm then treats the arena $X^{\geq J_2}$. By Lemma 16, it is enough to consider the region X^{J_2}. Let us explain how to compute $\lambda^2(v)$ from $\lambda^1(v)$ on this region. For $v = (v_7, \{2\})$, we have that $\lambda^2(v) = 1 + \min_{(v,v')\in E^X} \sup\{\mathrm{Cost}_1(\rho) \mid \rho \in \Lambda^1(v')\}$. As the unique successor of v is $(v_2, \{1, 2\})$, all λ^1-consistent plays beginning in this successor have cost 0. So, we have that $\lambda^2(v) = 1$. For the computation of $\lambda^2(v_6, \{2\})$, the same argument holds since $(v_6, \{2\})$ has the unique successor $(v_7, \{2\})$. The vertex $(v_1, \{2\})$ has two successors: $(v_6, \{2\})$ and $(v_3, \{2\})$. Again, we know that all λ^1-consistent plays beginning in $(v_6, \{2\})$ have cost 2. From $(v_3, \{2\})$ however, one can easily check that the play $(v_3, \{2\})(v_0, \{2\})((v_4, \{2\}))^\omega$ is λ^1-consistent and has cost $+\infty$ for player 1. Thus, we obtain that $\lambda^2(v_1, \{2\}) = 3$. For the other vertices of X^{J_2}, one can see that $\lambda^2(v) = \lambda^1(v)$.

Finally, we can check that the local fixed point is reached in the arena $X^{\geq J_2}$ (resp. $X^{\geq J_1}$) with $\lambda^3 = \lambda^2$ (resp. $\lambda^6 = \lambda^5 = \lambda^*$). Therefore the respective fixpoints are reached with $k^*_2 = 2$ and $k^*_1 = 5$. The labeling function indicated in Figure 2 is the one of λ^*. ◀

Table 1 The different steps of the algorithm computing λ^* for the extended game of Figure 2.

Region	Π	$\{2\}$							$\emptyset$					
	v	v_0	v_1	v_6	v_7	v_3	v_4	v_5	v_0	v_1	v_6	v_7	v_3	v_4
$\lambda^0 = \lambda^1$	0	0	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
$\lambda^2 = \lambda^3$	0	0	3	2	1	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
λ^4	0	0	3	2	1	$+\infty$	$+\infty$	$+\infty$	$+\infty$	3	2	1	$+\infty$	$+\infty$
$\lambda^5 = \lambda^*$	0	0	3	2	1	$+\infty$	$+\infty$	$+\infty$	4	3	2	1	$+\infty$	$+\infty$

4 Counter graph

In this section, given a labeling function λ, we introduce the concept of *counter graph* such that its infinite paths coincide with the plays that are λ-consistent. We then show that the counter graph associated with the fixpoint function λ^* computed by Algorithm 1 has an exponential size, an essential step to prove PSPACE membership of the constraint problem.

For the entire section, we fix a reachability game $\mathcal{G} = (G, (F_i)_{i\in\Pi}, (\mathrm{Cost}_i)_{i\in\Pi})$ and $\mathcal{X} = (X, (F_i^X)_{i\in\Pi}, (\mathrm{Cost}_i)_{i\in\Pi})$ its associated extended game.

A labeling function λ give constraints on costs of plays from each vertex in X, albeit only for the owner of this vertex. However, by the property of λ-consistence, constraints for a player carry over all the successive vertices, whether they belong to him or not. In order to check efficiently this property, we introduce the counter graph to keep track *explicitly* of the accumulation of constraints for all players at each step of a play. We first fix some notation.

▶ **Definition 19** (Maximal finite range). *Let $\lambda : V^X \to \mathbb{N} \cup \{+\infty\}$ be a labeling function.*

- *We consider* restrictions *of λ to sub-arenas of V^X as follows. Let $n \in \{1, \dots, N\}$, we denote by $\lambda_n : V^{J_n} \to \mathbb{N} \cup \{+\infty\}$ the restriction of λ to V^{J_n}. Similarly we denote by $\lambda_{\geq n}$ (resp. $\lambda_{>n}$) the restriction of λ to $V^{\geq J_n}$ (resp. $V^{>J_n}$).*
- *The* maximal finite range *of λ, denoted by $\mathrm{mR}(\lambda)$, is equal to $\mathrm{mR}(\lambda) = \max\{c \in \mathbb{N} \mid \lambda(v) = c$ for some $v \in V^X\}$ with the convention that $\mathrm{mR}(\lambda) = 0$ if λ is the constant function $+\infty$. We extend this notion to restrictions of λ with the convention that $\mathrm{mR}(\lambda_{>n}) = 0$ if J_n is a bottom region.*

Note that in the definition of maximal finite range, we only consider the *finite* values of λ.

▶ **Definition 20** (Counter Graph). *Let $\lambda : V^X \to \mathbb{N} \cup \{+\infty\}$ be a labeling function. Let $\mathcal{K} := \{0, \dots, K\} \cup \{+\infty\}$ with $K = \mathrm{mR}(\lambda)$. The* counter graph *$\mathbb{C}(\lambda)$ for $\mathcal{G}$ and λ is equal to $\mathbb{C}(\lambda) = (\Pi, V^C, (V_i^C)_{i\in\Pi}, E^C)$, such that:*

- $V^C = V^X \times \mathcal{K}^{|\Pi|}$
- $(v, (c_i)_{i\in\Pi}) \in V_j^C$ *if and only if* $v \in V_j^X$
- $((v, (c_i)_{i\in\Pi}), (v', (c_i')_{i\in\Pi})) \in E^C$ *if and only if:*
  - $(v, v') \in E^X$, *and*
  - *for every* $i \in \Pi$, $c_i' = \begin{cases} 0 & \text{if } i \in I(v') \\ c_i - 1 & \text{if } i \notin I(v'), v' \notin V_i^X \text{ and } c_i > 1 \\ \min(c_i - 1, \lambda(v')) & \text{if } i \notin I(v'), v' \in V_i^X \text{ and } c_i > 1. \end{cases}$

Intuitively, the counter graph is constructed such that once a value $\lambda(v)$ is finite for a vertex $v \in V_i^X$ along a play in $\mathcal{X}$, the corresponding path in $\mathbb{C}(\lambda)$ keeps track of the induced constraint by *(i)* decrementing the counter value c_i for the concerned player i by 1 at every step, *(ii)* updating this counter if a stronger constraint for player i is encountered by visiting a vertex v' with a smaller value $\lambda(v')$, and *(iii)* setting the counter c_i to 0 if player i has indeed reached his target set.

Note that there may be vertices with *no outgoing edges*. Indeed, consider a vertex $(v,(c_i)_{i\in\Pi}) \in V^C$ such that $c_j = 1$ for some player j. By construction of $\mathbb{C}(\lambda)$, the only outgoing edges from $(v,(c_i)_{i\in\Pi})$ must link to vertices $(v',(c'_i)_{i\in\Pi})$ such that $(v,v') \in E^X$, $c'_j = 0$ and $j \in I(v')$. However, it may be that no successor v' of v in X is such that $j \in I(v')$.

Note as well that for each vertex $v \in V^X$, there exist many different vertices $(v,(c_i)_{i\in\Pi})$ in $\mathbb{C}(\lambda)$, one for each counter values profile. However, the intended goal of $\mathbb{C}(\lambda)$ is to monitor explicitly the constraints accumulated by each player along a play in $\mathcal{X}$ regarding λ. Thus, we will only consider paths in $\mathbb{C}(\lambda)$ that start in vertices $(v,(c_i)_{i\in\Pi})$ such that the counter values correspond indeed to the constraint at the *beginning of a play* in $\mathcal{X}$ regarding λ:

▶ **Definition 21** (Starting vertex in $\mathbb{C}(\lambda)$). *Let $v \in V^X$. We distinguish one vertex $v^C = (v,(c_i)_{i\in\Pi})$ in V^C, such that for every $i \in \Pi$, the counter value c_i is equal to 0 if $i \in I(v)$, to $\lambda(v)$ if $i \notin I(v)$ and $v \in V_i^X$, and to $+\infty$ otherwise. We call v^C the* starting vertex *associated with v, and denote by $\mathrm{SV}(\lambda)$ the set of all starting vertices in $\mathbb{C}(\lambda)$.*

There exists a correspondence between λ-consistent plays in $\mathcal{X}$ and *infinite* paths from starting vertices in $\mathbb{C}(\lambda)$, called *valid paths*, in the following way. On one hand, every play ρ in $\mathcal{X}$ that is not λ-consistent does not appear in the counter graph: the first constraint regarding λ that is violated along ρ is reflected by a vertex in $\mathbb{C}(\lambda)$ with a counter value getting to 1 and no outgoing edges. On the other hand, λ-consistent plays in $\mathcal{X}$ have a corresponding infinite path in the counter graph $\mathbb{C}(\lambda)$. This is formalized in the next lemma:

▶ **Lemma 22.** *There exists a λ-consistent play $\rho = \rho_0\rho_1\dots$ in $\mathrm{Plays}_X(v)$ with $v \in V^X$ if, and only if there exists an associated infinite path $\pi = \pi_0\pi_1\dots$ in $\mathbb{C}(\lambda)$ such that $\pi_0 = v^C \in \mathrm{SV}(\lambda)$ and ρ is the projection of π on V^X (π_n is of the form $(\rho_n,(c'_i)_{i\in\Pi})$ for all $n \in \mathbb{N}$).*

Since the edge relation E^C in $\mathbb{C}(\lambda)$ respects the edge relation E^X in $\mathcal{X}$, the region decomposition of a path in $\mathcal{X}$ given in Lemma 8 can also be applied to a path in $\mathbb{C}(\lambda)$.

In order to prove the PSPACE membership for the constraint problem, we need to show that the counter graph $\mathbb{C}(\lambda^*)$, with λ^* the fixpoint function computed by Algorithm 1, has an exponential size. To this end it is enough to show an exponential upper bound on the maximal finite range $\mathrm{mR}(\lambda^*)(= \mathrm{mR}(\lambda^{k_1^*}_{\geq 1}))$ of λ^*. To do so, we prove with Theorem 23, by induction on the number of computation steps k of Algorithm 1, an exponential upper bound on $\mathrm{mR}(\lambda^k_\ell)$ and $\mathrm{mR}(\lambda^k_{\geq\ell})$ for every region X^{J_ℓ} and every step k.

▶ **Theorem 23.** *For every $k \in \mathbb{N}$ and region X^{J_ℓ}, we have*

$$\mathrm{mR}(\lambda^k_\ell) \leq \mathcal{O}(|V|^{(|V|+3)\cdot(|\Pi|+2)}) \quad \textit{and} \quad \mathrm{mR}(\lambda^k_{\geq\ell}) \leq \mathcal{O}(|V|^{(|V|+3)\cdot(|\Pi|+2)}).$$

The next corollary will be useful in Section 5 where not only the maximal finite range must be exponentially bounded, but also the cost of any play satisfying the current constraints.

▶ **Corollary 24.** *Let $v \in V^X$ with $I(v) = J_\ell$ with $\ell < N$. Let $k \in \mathbb{N}$ such that $k > k^*_{\ell+1}$. Suppose there exists $c \in \mathbb{N}$ such that $\sup\,\{\mathrm{Cost}_i(\rho) \mid \rho \in \Lambda^k(v)\} = c$. Then, we have:*

$$c \leq \mathcal{O}(|V|^{(|V|+3)\cdot(|\Pi|+2)}).$$

Let us give a few ingredients of the induction of Theorem 23. For each region X^{J_ℓ}, the value $\mathrm{mR}(\lambda^0_\ell)$ is always equal to 0. For the general case, the value $\mathrm{mR}(\lambda^{k+1}_\ell)$ depends on the values of λ^{k+1}_ℓ, which, when finite, are in turn determined by the maximal cost of the λ^k-consistent plays starting in region X^{J_ℓ} (see Definition 13). A crucial point to obtain these bounds is that this maximal cost corresponds to the length of the longest cycle-free prefix of an infinite path starting in region X^{J_ℓ} in $\mathbb{C}(\lambda^k)$. Furthermore, we can evaluate this maximal length in terms of the values of $\text{mR}(\lambda^k_\ell)$ and $\text{mR}(\lambda^k_{>\ell})$, which are bounded by the previous step of the induction. The following key technical lemma formalizes this idea.

▶ **Lemma 25.** *Let v^C be a starting vertex in $\text{SV}(\lambda)$ associated with $v \in V^X$ such that $I(v) = J_\ell$. Let π be a finite prefix of a valid path in $\mathbb{C}(\lambda)$ such that $\pi_0 = v^C$ and π does not contain any cycle. Then*

$$|\pi| \leq |V| + 2 \cdot \text{mR}(\lambda_\ell) + \sum_{r=|J_\ell|+1}^{|\Pi|} |V| + 2 \cdot \max_{\substack{J_j > J_\ell \\ |J_j| = r}} \text{mR}(\lambda_j).$$

Proof sketch. Let π be a finite prefix of a valid path in $\mathbb{C}(\lambda)$ as in the statement. Let $\pi[\ell] \dots \pi[m]$ be its region decomposition according to Lemma 8, graphically represented in Figure 3. Let ρ be the corresponding path in $\mathcal{X}$ and $\rho[\ell] \dots \rho[m]$ be its region decomposition. Let us consider a fixed non-empty section $\pi[n]$.

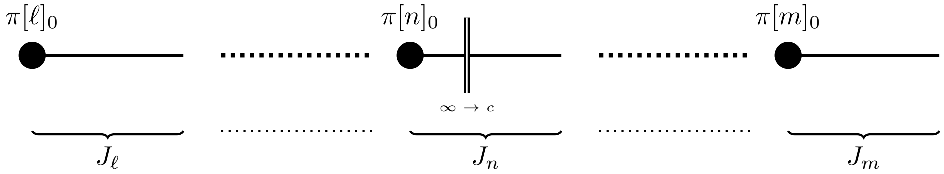


Figure 3 Region decomposition of π.

Suppose first that the counter values at $\pi[n]_0$ are either 0 or $+\infty$. Let us prove that along $\pi[n]$, there can be at most $|V|$ steps before reaching a vertex with a finite positive value of λ:

- assume there is a cycle in the corresponding section $\rho[n]$ in $\mathcal{X}$ such that from $\rho[n]_0$ and along the cycle, all the values of λ are either 0 or $+\infty$,
- by construction of $\mathbb{C}(\lambda)$, the counter values in the corresponding prefix of $\pi[n]$ remain fixed for each vertex of this prefix: as no value of λ is positive and finite, no counter value can be decremented,
- thus, the cycle in $\rho[n]$ is also a cycle in $\pi[n]$ which is impossible by hypothesis,
- thus there is no such cycle in $\rho[n]$, and as there are at most $|V|$ vertices in region X^{J_n}, $\rho[n]$ can have a prefix of length at most $|V|$ with only values 0 or $+\infty$ for λ, implying that this is also the case for $\pi[n]$.

Therefore, we can decompose $\pi[n]$ into a (possibly empty) prefix of length at most $|V|$, and a (possibly empty) suffix where at least one counter value c'_i, for some i, is a positive finite value in its first vertex v'. This frontier between prefix and suffix of $\pi[n]$ is represented by a vertical double bar $\|$ with caption $\infty \to c$ in Figure 3. This value c'_i is bounded by $\text{mR}(\lambda_n)$, the maximal finite range of λ_n. From there, as the corresponding ρ is λ-consistent, player i reaches his target set in at most c'_i steps, and ρ enters a new region, which means that the section $\pi[n]$ is over. So, in that case, the length of $\pi[n]$ can be bounded by $|V| + \text{mR}(\lambda_n)$.

Suppose now that at vertex $\pi[n]_0$, there exists a counter value c_i for some player i that is neither 0 nor $+\infty$. This means that there was a constraint for player i initialized in a previous section $\pi[n']$, with $n' < n$, that has carried over to $\pi[n]_0$, via decrements of at least 1 per step. We know that the initial finite counter value is bounded by $\text{mR}(\lambda_{n'})$, and appeared before the end of section $\pi[n']$. Thus the length from the end of section $\pi[n']$ to the end of section $\pi[n]$ is bounded by $\text{mR}(\lambda_{n'})$, as again, once the counter value attains 0 for player i, the path π has entered the next section.

Therefore, considering the possible cases for each section, we can bound the total length of π as follows: $|\pi| \leq \sum_{j=\ell}^{m} |V| + 2 \cdot \mathrm{mR}(\lambda_j)$.

Finally, remark that by I-monotonicity, it is actually the case that only (and at most) $|\Pi|$ different non-empty sections can appear in the decomposition of π. Furthermore, for each $n \in \{\ell+1, \ldots, N\}$, we have $\mathrm{mR}(\lambda_n) \leq \max\{\mathrm{mR}(\lambda_j) \mid J_j > J_\ell,\ |J_j| = |J_n|\}$ by Definition 19. Thus, we obtain the bound stated in Lemma 25. ◀

5 PSPACE completeness

In this section, we prove that the constraint problem is PSPACE-complete (Theorem 4). Given a reachability game $(\mathcal{G}, v_0)$ and two thresholds $x, y \in (\mathbb{N} \cup \{+\infty\})^{|\Pi|}$, this problem is to decide whether there exists an SPE σ in this game such that for all $i \in \Pi$, $x_i \leq \mathrm{Cost}_i(\langle\sigma\rangle_{v_0}) \leq y_i$. As the proof of the PSPACE-hardness (based on a reduction from the QBF problem) is nearly the same as for qualitative reachability games [10], we only prove the PSPACE-easyness.

▶ **Proposition 26.** *The constraint problem for quantitative reachability games is in PSPACE.*

Let us provide a high level sketch of the proof of our PSPACE procedure. Thanks to Theorem 17, solving the constraint problem reduces in finding a λ^*-consistent play ρ in $(\mathcal{X}, x_0)$ satisfying the constraints. By Lemma 22, the latter problem reduces in finding a corresponding valid path π in the counter graph $\mathbb{C}(\lambda^*)$, satisfying the constraints. Roughly speaking, in order to solve our initial problem, it suffices to decide the existence of a valid path that is a lasso and satisfies the constraints in the counter graph, which is exponential in the size of our original input. Classical arguments, using Savitch's Theorem can thus be used to prove the PSPACE membership. Nevertheless, the detailed proof is more intricate for two reasons. The first reason is that the counter graph is based on the labeling function λ^*. We thus also have to prove that this function λ^* can be computed in PSPACE. The second reason is that, a priori, although we know that the counter graph is of exponential size, we do not know explicitly its size. This is problematic when using classical NPSPACE algorithms that guess some path in a graph of exponential size, where a counter bounded by the size of the graph is needed to guarantee the termination of the procedure. In order to overcome this, we also need a PSPACE procedure to obtain the actual size of the counter graph. Recall that, as $|\mathbb{C}(\lambda^*)| = |V| \cdot 2^{|\Pi|} \cdot (K^* + 1)^{|\Pi|}$ with $K^* = \mathrm{mR}(\lambda^*)$, we only have to determine the value of K^*. This is possible thanks to Proposition 27.

▶ **Proposition 27.** *Given a quantitative game $(\mathcal{G}, v_0)$, for all $k \in \mathbb{N}$, for all J_ℓ with $\ell \in \{1, \ldots, N\}$, the set $\{\lambda^k(v) \mid v \in V^{J_\ell}\}$ and the value $\mathrm{mR}(\lambda^k_{\geq \ell})$ can be computed in PSPACE.*

Proof sketch. The whole procedure works by induction on k, the steps in the computation of the labeling function λ^*. Moreover, it exploits the structural evolution of the local fixpoints formalized in Proposition 15: once a step is fixed, we proceed region by region, beginning with the bottom region J_N and then proceeding bottom-up by following the total order on $\mathcal{I}$.

Let X^{J_ℓ} be a region, we aim at proving that $\{\lambda^{k+1}(v) \mid v \in V^{J_\ell}\}$ and $\mathrm{mR}(\lambda^{k+1}_{\geq \ell})$ are computable in PSPACE, assuming that *(i)* we know $\{\lambda^k(v) \mid v \in V^{J_\ell}\}$, and *(ii)* we can compute $\{\lambda^k(v) \mid v \in V^{J'}\}$ for all $J' > J_\ell$ and $\mathrm{mR}(\lambda^k_{\geq \ell})$ in PSPACE.

We first focus on the computation of λ^{k+1} from λ^k. Let k^*_ℓ (resp. $k^*_{\ell+1}$) be the step where the fixpoint is reached for region X^{J_ℓ} (resp. $X^{J_{\ell+1}}$). Recall that $k^*_{\ell+1} < k^*_\ell$. When $k \leq k^*_{\ell+1}$ (resp. $k > k^*_\ell$), we have that $\lambda^{k+1}(v) = \lambda^k(v)$, for each $v \in V^{J_\ell}$, by Lemma 16. The tricky case in when $k^*_{\ell+1} < k \leq k^*_\ell$[3], let us focus on it.

[3] Notice the little difference with the inequalities given in Lemma 16. When $k = k^*_\ell$, we still need to compute λ^{k+1} to realize that the fixpoint is effectively reached.

In this case, the computation of $\lambda^{k+1}(v)$ from $\lambda^k(v)$, for all $v \in V^{J_n}$, relies on the maximum of the cost of the λ^k-consistent plays (see Definition 13). Computing this maximum is equivalent to guess the existence of lassoes with a certain cost in the counter graph $\mathbb{C}(\lambda^k_{\geq \ell})$. The size of these lassoes depends on the size of $\mathbb{C}(\lambda^k_{\geq \ell})$ equal to $|V| \cdot 2^{|\Pi|} \cdot (K+1)^{|\Pi|}$ where $K = \mathrm{mR}(\lambda^k_{\geq \ell})$. Moreover, each lasso is guessed region by region. As we can compute in PSPACE the value K and the sets $\{\lambda^k(v) \mid v \in V^{J'}\}$ for all $J' \geq J_\ell$ by induction hypothesis, we can also compute in PSPACE the maximum of the cost of the λ^k-consistent plays and thus the set $\{\lambda^{k+1}(v) \mid v \in V^{J_\ell}\}$. The correctness of this approach relies on Proposition 15 which ensures that the values of λ^* have already reached a fixpoint for all the $v \in V^{J'}$ such that $J' > J_\ell$. Once we have computed $\{\lambda^{k+1}(v) \mid v \in V^{J_\ell}\}$ in PSPACE, it is clear that $\mathrm{mR}(\lambda^{k+1}_{\geq \ell})$ can also be computed in PSPACE because $\mathrm{mR}(\lambda^{k+1}_{\geq \ell}) = \max\{\mathrm{mR}(\lambda^{k+1}_{\ell}), \mathrm{mR}(\lambda^{k+1}_{\geq \ell+1})\}$ (notice that $\mathrm{mR}(\lambda^{k+1}_{\geq \ell+1}) = \mathrm{mR}(\lambda^{k}_{\geq \ell+1})$ because $k > k^*_{\ell+1}$ and thus $X^{J_{\ell+1}}$ has already reached its local fixpoint and so the induction hypothesis can be used).

Notice that all these arguments hold because: *(i)* the value of $\mathrm{mR}(\lambda^k_{\geq \ell})$ and any value of $\lambda^k(v)$ with $v \in V^{\geq \ell}$ (or of $\lambda^{k+1}(v)$ with $v \in V^{\ell}$) can be encoded in polynomial size memory (by Theorem 23); *(ii)* the value of the maximal cost of λ^k-consistent plays is at most exponential in the inputs (by Corollary 24); *(iii)* any set $\{\lambda^k(v) \mid v \in V^{J'}\}$, $J' \geq J_\ell$ (or the set $\{\lambda^{k+1}(v) \mid v \in V^{J_\ell}\}$) is composed of at most $|V|$ values which can be encoded with polynomial size memory by *(i)*, and *(iv)* we have a polynomial number of such values to keep in memory. ◀

Proof of Proposition 26. Let $(\mathcal{G}, v_0)$ be an initialized reachability game and let $x, y \in (\mathbb{N} \cup \{+\infty\})^{|\Pi|}$ be two thresholds. Let $(\mathcal{X}, x_0)$ be its extended game with $x_0 = (v_0, I_0)$. Let $\mathbb{C}(\lambda^*_{\geq I_0})$ be the counter graph restricted to the arena $X^{\geq I_0}$ and so $|\mathbb{C}(\lambda^*_{\geq I_0})| = |V| \cdot 2^{|\Pi|} \cdot (K^0+1)^{|\Pi|}$ with $K^0 = \mathrm{mR}(\lambda^*_{\geq I_0})$. Let us recall that K^0 can be computed (resp. encoded) in PSPACE thanks to Proposition 27 (resp. Theorem 23).

We have to guess a lasso $\pi = hg^\omega$ in the counter graph $\mathbb{C}(\lambda^*_{\geq I_0})$ such that its corresponding play ρ^X in the extended game satisfies the constraints, *i.e.*, $x_i \leq \mathrm{Cost}_i(\rho^X) \leq y_i$ for all $i \in \Pi$. The length L of π is at most $\max(\max\{y_i \mid y_i < +\infty\}, |\mathbb{C}(\lambda^*_{\geq I_0})|) + 2 \cdot |\mathbb{C}(\lambda^*_{\geq I_0})|$ as the cycle of π can appear after the constraints $y_i < +\infty$ are satisfied.

In order to have a PSPACE procedure, we cannot guess π entirely and we have to proceed region by region. If $I_0 = J_\ell$ for some $\ell \in \{1, \dots, N\}$, we consider the region decomposition $\pi[\ell]\pi[\ell+1]\dots\pi[t]$ of π, with $t \in \{\ell, \dots, N\}$, where some sections $\pi[m]$ may be empty.

We first guess the first vertex of the cycle g and then we guess successively $\pi[\ell]\pi[\ell+1]$ and so on. To guess $\pi[m]$ with $\ell \in \{m, \dots, t\}$, assuming it is not empty, we guess one by one its vertices. To guess a vertex $(v', J_m, (c'_i)_{i \in \Pi})$ we only have to keep its predecessor in memory and to know the value $\lambda^*(v', J_m)$. So, we need to know $\{\lambda^*(v) \mid v \in V^{J_m}\}$. Thanks to Proposition 27, we can obtain this set in polynomial size memory and once we move to another region, we can forget this set and compute the new one.

Additionally, we have a counter C_L to count the length of π and for each player $i \in \Pi$ we have a counter C_i keeping track the current cost of player i along π. For all $i \in \Pi$, we have that $C_i, C_L \leq L$. Henceforth, all these counters can be encoded with polynomial size memory. Moreover, in addition to these counters, we currently maintain: *(i)* the first vertex of g, *(ii)* the current vertex we are guessing, *(iii)* its predecessor and, *(iv)* the set $\{\lambda^*(v) \mid v \in V^{J_m}\}$ (only $|V|$ values which can be encoded in polynomial size memory by Theorem 23) if we are guessing $\pi[m]$. As for *(ii)* and *(iii)* a vertex in the counter graph is composed of a vertex of V, a subset I of Π and $|\Pi|$ counter values which are at most exponential in the input (Theorem 23), all this procedure can be done in PSPACE. ◀

References

1 Nicolas Basset, Ismaël Jecker, Arno Pauly, Jean-François Raskin, and Marie van den Bogaard. Beyond Admissibility: Dominance Between Chains of Strategies. In *CSL*, volume 119 of *LIPIcs*, pages 10:1–10:22. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2018.

2 Dietmar Berwanger. Admissibility in Infinite Games. In *STACS*, volume 4393 of *LNCS*, pages 188–199. Springer, 2007.

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 *LATA*, volume 9618 of *LNCS*, pages 3–23. Springer, 2016.

4 Romain Brenguier, Arno Pauly, Jean-François Raskin, and Ocan Sankur. Admissibility in Games with Imperfect Information (Invited Talk). In *CONCUR*, volume 85 of *LIPIcs*, pages 2:1–2:23. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

5 Romain Brenguier, Guillermo A. Pérez, Jean-Francois Raskin, and Ocan Sankur. Admissibility in Quantitative Graph Games. In *FSTTCS*, volume 65 of *LIPIcs*, pages 42:1–42:14, 2016.

6 Romain Brenguier, Jean-François Raskin, and Ocan Sankur. Assume-admissible synthesis. In *CONCUR*, LIPIcs 42, pages 100–113. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.

7 Romain Brenguier, Jean-François Raskin, and Mathieu Sassolas. The complexity of admissibility in Omega-regular games. In *CSL-LICS*, pages 23:1–23:10. ACM, 2014.

8 Thomas Brihaye, Véronique Bruyère, Julie De Pril, and Hugo Gimbert. On Subgame Perfection in Quantitative Reachability Games. *Logical Methods in Computer Science*, 9(1), 2012.

9 Thomas Brihaye, Véronique Bruyère, Aline Goeminne, and Jean-François Raskin. Constrained Existence Problem for Weak Subgame Perfect Equilibria with ω-Regular Boolean Objectives. In *GandALF*, pages 16–29, 2018.

10 Thomas Brihaye, Véronique Bruyère, Aline Goeminne, and Jean-François Raskin. Constrained existence problem for weak subgame perfect equilibria with omega-regular Boolean objectives. *CoRR*, abs/1806.05544, 2018.

11 Thomas Brihaye, Véronique Bruyère, Noémie Meunier, and Jean-François Raskin. Weak Subgame Perfect Equilibria and their Application to Quantitative Reachability. In *CSL*, volume 41 of *LIPIcs*, pages 504–518. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

12 Thomas Brihaye, Julie De Pril, and Sven Schewe. Multiplayer Cost Games with Simple Nash Equilibria. In *LFCS*, volume 7734 of *Lecture Notes in Computer Science*, pages 59–73. Springer, 2013.

13 Véronique Bruyère. Computer Aided Synthesis: A Game-Theoretic Approach. In *DLT*, volume 10396 of *LNCS*, pages 3–35. Springer, 2017.

14 Véronique Bruyère, Stéphane Le Roux, Arno Pauly, and Jean-François Raskin. On the Existence of Weak Subgame Perfect Equilibria. In *FOSSACS*, volume 10203 of *LNCS*, pages 145–161, 2017.

15 K. Chatterjee, T. A. Henzinger, and M. Jurdzinski. Games with secure equilibria. *Theoretical Computer Science*, 365:67–82, 2006.

16 Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Strategy logic. *Inf. Comput.*, 208(6):677–693, 2010.

17 Rodica Condurache, Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. The Complexity of Rational Synthesis. In *ICALP*, volume 55 of *LIPIcs*, pages 121:1–121:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

18 Emmanuel Filiot, Raffaella Gentilini, and Jean-François Raskin. Rational Synthesis Under Imperfect Information. In *LICS*, pages 422–431. ACM, 2018.

19 Dana Fisman, Orna Kupferman, and Yoad Lustig. Rational Synthesis. In *TACAS*, volume 6015 of *LNCS*, pages 190–204. Springer, 2010.

20 János Flesch, Jeroen Kuipers, Ayala Mashiah-Yaakovi, Gijs Schoenmakers, Eilon Solan, and Koos Vrieze. Perfect-Information Games with Lower-Semicontinuous Payoffs. *Mathematics of Operation Research*, 35:742–755, 2010.

21 Erich Grädel and Michael Ummels. Solution Concepts and Algorithms for Infinite Multiplayer Games. In *New Perspectives on Games and Interaction*, volume 4, pages 151–178. Amsterdam University Press, 2008.

22 Orna Kupferman, Giuseppe Perelli, and Moshe Y. Vardi. Synthesis with Rational Environments. In *EUMAS*, LNCS 8953, pages 219–235. Springer, 2014.

23 Fabio Mogavero, Aniello Murano, and Moshe Y. Vardi. Reasoning About Strategies. In *FSTTCS 2010*, volume 8 of *LIPIcs*, pages 133–144. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2010.

24 J. F. Nash. Equilibrium points in n-person games. In *PNAS*, volume 36, pages 48–49. National Academy of Sciences, 1950.

25 Martin J. Osborne. *An introduction to game theory.* Oxford Univ. Press, 2004.

26 A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *POPL*, pages 179–190. ACM Press, 1989.

27 Michael Ummels. Rational Behaviour and Strategy Construction in Infinite Multiplayer Games. In *FSTTCS*, volume 4337 of *LNCS*, pages 212–223. Springer, 2006.

28 Michael Ummels. The Complexity of Nash Equilibria in Infinite Multiplayer Games. In *FOSSACS*, volume 4962 of *LNCS*, pages 20–34. Springer, 2008.

29 Michael Ummels and Dominik Wojtczak. The Complexity of Nash Equilibria in Limit-Average Games. In *CONCUR*, volume 6901 of *LNCS*, pages 482–496. Springer, 2011.

On the Complexity of Reachability in Parametric Markov Decision Processes

Tobias Winkler
RWTH Aachen University, Germany
tobias.winkler1@rwth-aachen.de

Sebastian Junges
RWTH Aachen University, Germany
sebastian.junges@cs.rwth-aachen.de

Guillermo A. Pérez
University of Antwerp, Belgium
guillermoalberto.perez@uantwerpen.be

Joost-Pieter Katoen
RWTH Aachen University, Germany
katoen@cs.rwth-aachen.de

Abstract

This paper studies parametric Markov decision processes (pMDPs), an extension to Markov decision processes (MDPs) where transitions probabilities are described by polynomials over a finite set of parameters. Fixing values for all parameters yields MDPs. In particular, this paper studies the complexity of finding values for these parameters such that the induced MDP satisfies some reachability constraints. We discuss different variants depending on the comparison operator in the constraints and the domain of the parameter values. We improve all known lower bounds for this problem, and notably provide ETR-completeness results for distinct variants of this problem. Furthermore, we provide insights in the functions describing the induced reachability probabilities, and how pMDPs generalise concurrent stochastic reachability games.

2012 ACM Subject Classification Theory of computation → Probabilistic computation; Theory of computation → Logic and verification; Theory of computation → Markov decision processes

Keywords and phrases Parametric Markov decision processes, Formal verification, ETR, Complexity

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.14

Related Version `https://arxiv.org/abs/1904.01503`

Funding This work has been supported by the DFG RTG 2236 “UnRAVeL”, and the ERC Advanced Grant 787914 “FRAPPANT”.

Acknowledgements We would like to thank Krishnendu Chatterjee for his pointer to CSRGs.

1 Introduction

Markov decision processes (MDPs) are *the* model to reason about sequential processes under (stochastic) uncertainty and non-determinism. Markov chains (MCs) are MDPs without non-determinism. Often, probability distributions in these models are difficult to assess precisely during design time of a system. This shortcoming has led to interval MCs [15, 35, 50, 54] and interval MDPs (also known as Bounded-parameter MDPs) [27, 42, 58], which allow for interval-labelled transitions. Analysis under interval Markov models is often too pessimistic: The actual probabilities on the transitions are considered to be non-deterministically and *locally* chosen. Intuitively, consider the probability of a coin-flip yielding heads in some stochastic environment. In interval models, the probability may vary with the local memory state of an agent acting in this environment. Such behaviour is unrealistic. *Parametric*

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 14; pp. 14:1–14:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

MCs/MDPs [19, 23, 28, 39] (pMCs, pMDPs) overcome this limitation by adding dependencies (or couplings) between various transitions – they add global restrictions to the selection of the probability distributions. Intuitively, the probability of flipping heads can be arbitrary, but should be independent of an agent's local memory. Such couplings are similar to restrictions on schedulers in decentralised/partially observable MDPs, considered in e.g., [5, 26, 51].

Technically, pMDPs label their transitions with polynomials over a finite set of parameters. Fixing all parameter values yields MDPs. The synthesis problem considered in this paper asks to find parameter values such that the induced MDPs satisfy reachability constraints. Such reachability constraints state that the probability – under some/all possible ways to resolve non-determinism in the MDP – to reach a target state is (strictly) above or below a threshold. A sample synthesis problem is thus: "Are there parameter values such that for all possible ways to resolve the non-determinism, the probability to reach a target state exceeds $\frac{1}{2}$?" Variants of the synthesis problem are obtained by varying the reachability constraints, and the domain of the parameter values. Parameter synthesis is supported by the model checkers PRISM [38] and Storm [22], and dedicated tools PARAM [29] and PROPhESY [21]. The complexity of the decision problems corresponding to parameter synthesis is mostly open.

This paper significantly extends complexity results for parameter synthesis in pMCs and pMDPs. Table 1 on page 5 gives an overview of new results: Most prominently, it establishes ETR-completeness of reachability problems for pMCs with non-strict comparison operators, and establishes NP-hardness for pMCs with strict comparison operators. For pMDPs with demonic non-determinism, it establishes ETR-completeness for any comparison operator. For angelic non-determinism, mostly the synthesis problems are equivalent to their pMC counterparts. When considering pMDPs with a fixed number of variables, we establish uniform NP upper bounds for parameter synthesis under angelic or demonic non-determinism. These results are partially based on properties of pMDPs scattered in earlier work, and use a strong connection between polynomial inequalities and parameter synthesis.

Finally, pMDPs are interesting generalisations of other models: [37] shows that parameter synthesis in pMCs is equivalent to the synthesis of finite-state controllers (with a-priori fixed bounds) of partially observable MDPs (POMDPs) [46] under reachability constraints. Thus, as a side product we improve complexity bounds [10, 56] for (a-priori fixed) memory bounded strategies in POMDPs. In this paper, we show how pMDPs generalise concurrent stochastic reachability games [12, 20, 52]. We finish the paper by drawing some connections with robust schedulers, i.e. the question of how to optimally resolve non-determinism taking into account the uncertainty in the stochastic dynamics. Proofs are given in the related technical report.

Related work. Various results in this paper extend work by Chonev [16], who studied a model of augmented interval Markov chains. These coincide with parametric Markov chains. The work also builds upon results by Hutschenreiter *et al.* [33], in particular upon the result that pMCs with an a-priori fixed number of parameters can be checked in P. Furthermore, they study the complexity of PCTL model checking of pMCs. The complexity of finite-state controller synthesis in POMDPs has been studied in [10, 56]. Some of the proofs for ETR-completeness presented here reuse ideas from [48].

Methods (and implementations) to analyse pMCs by computing their characteristic *solution function* are considered in [19, 21, 23–25, 29, 33, 34]. Sampling-based approaches to find feasible instantiations in pMDPs are considered by [14, 28], while [3, 18] utilise optimisation methods. Finally, [44] presents a method to prove the absence of solutions in pMDPs by iteratively considering simple stochastic games [17]. Some other works on Markov models with structurally equivalent yet parameterised dynamics include [8, 9, 13, 53]. Parameter synthesis with statistical guarantees has been explored in, e.g., [6]. Further work on parameter synthesis in Markov models has been surveyed in [36].

2 Preliminaries

Let X be a finite set of variables. Let $\mathbb{Q}[X]$ and $\mathbb{Q}(X)$ denote the set of all rational-coefficient polynomial and rational functions on X, respectively. A rational function f/g can be represented as a pair (f, g) of polynomials. In turn, a polynomial can be represented as a sum of terms, where each term is given by a *coefficient* and a *monomial.* The (total) degree of a polynomial is the maximum over the sum of the exponents in the monomials. A polynomial is quadratic (respectively, quadric), if its total degree is two (four) or less. For a rational function $f(x_1, \ldots, x_k) \in \mathbb{Q}(X)$ and an instantiation $val\colon X \to \mathbb{R}$ we write $f[val]$ for the value $f(val(x_1), \ldots, val(x_k))$. We use $\bowtie$ to denote either of $\{\leq, <, \geq, >\}$ and $\trianglerighteq$ for either $\{\geq, >\}$ (and $\trianglelefteq$ analogously). With $\overline{\bowtie}$, we denote the complement, e.g. $\overline{\leq} = >$.

Consider a finite set S. Let $\mathrm{Distr}(S)$ denote the set of all distributions over S, and $\mathrm{supp}(\delta) \subseteq S$ the support $\{s \in S \mid \delta(s) > 0\}$ of distribution $\delta \in \mathrm{Distr}(S)$.

2.1 Parametric Markov models

▶ **Definition 1** (pMDP). *A* parametric Markov Decision Process $\mathcal{M}$ *is a tuple* (S, X, Act, s_ι, P) *with* S *a (finite) set of* states, X *a finite set of* parameters, Act *a finite set of* actions, $s_\iota \in S$ *the initial state, and* $P\colon S \times Act \times S \to \mathbb{Q}[X] \cup \mathbb{R}$ *the probabilistic transition function.*

Parameter-free pMDPs coincide with standard MDPs, as in [43]. We define $Act(s) = \{a \in Act \mid \exists s' \in S.\ P(s, a, s') \neq 0\}$. If $|Act(s)| = 1$ for all $s \in S$, then $\mathcal{M}$ is a *parametric Markov chain* (pMC). We denote its transitions with $P(s, s')$ and omit the actions.

A pMDP is *simple* if and only if non-constant probabilities labelling transitions (s, a, s') are of the form x or $1 - x$, and the sum of outgoing transitions from a state-action pair always is (equivalent to) 1. Formally, simple pMDPs satisfy the following two properties:

- $P(s, a, s') \in \{x, 1 - x \mid x \in X\} \cup \mathbb{R}$ for all $s, s' \in S$ and $a \in Act$; and
- $\sum_{s' \in S} P(s, a, s') = 1$ for all $s \in S$ and $a \in Act(s)$.

▶ **Definition 2** (Instantiation). *Let* $\mathcal{M} = (S, X, Act, s_\iota, P)$ *be a pMDP. An instantiation* $val\colon X \to \mathbb{R}$ *is* well-defined *if the induced functions* $P(s, a, \cdot)$ *are distributions over* S*, i.e.*

$$\forall s, s' \in S, \forall a \in Act.\ 0 \leq P(s, a, s')[val] \in \mathbb{R} \wedge \sum_{\hat{s} \in S} P(s, a, \hat{s})[val] = 1.$$

Let $\mathcal{M}[val]$ denote the parameter-free MDP in which $P(s, a, s')$ has been replaced by $P(s, a, s')[val]$. We denote with $P^{val}_{=0} := \{(s, a, s') \in S \times Act \times S \mid P(s, a, s') \neq 0 \wedge P(s, a, s')[val] = 0\}$ the transitions of $\mathcal{M}$ that become 0 in $\mathcal{M}[val]$. A well-defined instantiation val is *graph-preserving* if the topology of the pMDP is preserved, i.e. if $P^{val}_{=0} = \emptyset$.

The (well-defined) parameter space $\mathcal{P}^{\mathrm{wd}}_{\mathcal{M}}$ for $\mathcal{M}$ is $\{val\colon X \to \mathbb{R} \mid val \text{ is well defined}\}$ and the graph-preserving parameter space $\mathcal{P}^{\mathrm{gp}}_{\mathcal{M}} := \{val\colon X \to \mathbb{R} \mid val \text{ is graph-preserving}\}$. In simple pMDPs, the well-defined (respectively, graph-preserving) parameter space is the set of instantiations $val\colon X \to [0, 1]$ (respectively, $val\colon X \to (0, 1)$). We omit the subscript from $\mathcal{P}^{\mathrm{gp}}_{\mathcal{M}}$ and $\mathcal{P}^{\mathrm{wd}}_{\mathcal{M}}$ when the pMDP $\mathcal{M}$ is understood from the context.

A *graph-consistent region* R for $\mathcal{M}$ is a subset of $\mathcal{P}^{\mathrm{wd}}$ such that all instantiations in R induce the same graph, i.e., $P^{val}_{=0} = P^{val'}_{=0}$ for all $val, val' \in R$.

▶ Remark 3. For any simple pMDP, $\mathcal{P}^{\mathrm{wd}}$ can be partitioned into $3^{|X|}$ many graph-consistent regions R. For any graph-consistent region, we can (in linear time) construct a simple pMDP $\mathcal{M}'$ such that the graph-consistent region R corresponds to $\mathcal{P}^{\mathrm{gp}}_{\mathcal{M}'}$. Essentially, the construction merely removes the transitions $P^{val}_{=0}$ for $val \in R$, and adjusts the probabilities of some other transitions to 1 to ensure simplicity. A complete construction is given in [36].

Reachability, schedulers and induced Markov chains. Consider a parameter-free MC $\mathcal{M}$ and a state s_0. A *run* of $\mathcal{M}$ from s_0 is an infinite sequence of states $s_0 s_1 \dots$ such that $P(s_i, s_{i+1}) > 0$ for all $i \geq 0$. We denote by Runs^{s_0} the set of all runs of $\mathcal{M}$ that start with the state s_0. The *probability* of a measurable *event* $E \subseteq \mathrm{Runs}^{s_0}$ is defined using a standard cylinder construction [2, 43]. Let $\Pr_{\mathcal{M}}(\Diamond T)$ denote the probability to *eventually reach* T from the initial state of $\mathcal{M}$; and $\Pr_{\mathcal{M}}(s \to \Diamond T)$ denote the probability to eventually reach T starting from state s. We omit the subscript $\mathcal{M}$ if it is clear from the context.

To define reachability in pMDPs, we need to eliminate the non-determinism. We do so by means of a *scheduler* (a.k.a. a *policy* or *strategy*).

▶ **Definition 4** (Scheduler). *A randomised (memoryless) scheduler is a function $\sigma\colon S \to \mathrm{Distr}(Act)$ s.t. $\mathrm{supp}(\sigma(s)) \subseteq Act(s)$. A scheduler is deterministic if $|\mathrm{supp}(\sigma(s))| = 1$ (i.e. $\sigma(s)$ is Dirac) for every $s \in S$. We refer to deterministic schedulers as* schedulers*.*

We denote the set of randomised schedulers with $R\Sigma$, and (deterministic) schedulers with Σ.

For pMDP $\mathcal{M} = (S, X, Act, s_\iota, P)$ and $\sigma \in R\Sigma_{\mathcal{M}}$, the *induced pMC* $\mathcal{M}^\sigma$ is defined as (S, X, s_ι, P') with $P'(s, s') = \sum_{a \in Act} \sigma(s)(a) \cdot P(s, a, s')$. For simple pMDPs, the induced pMC of a deterministic scheduler is simple. Under randomised schedulers, the induced pMC can be transformed into a simple pMC (e.g. [37]). We abbreviate $\Pr_{\mathcal{M}^\sigma}$ by $\Pr^\sigma_{\mathcal{M}}$.

▶ Remark 5. Deterministic schedulers dominate randomised schedulers for reachability properties [43], i.e. for each MDP there exists a deterministic scheduler σ s.t. $\Pr^\sigma_{\mathcal{M}}(\Diamond T) = \sup_{\sigma' \in R\Sigma} \Pr^{\sigma'}_{\mathcal{M}}(\Diamond T)$. Therefore, in the remainder, we focus on deterministic schedulers.

▶ **Definition 6** (Solution function). *For a pMC $\mathcal{M}$ and a state s, let the* solution function *$sol^{\mathcal{M},T}_s \colon \mathcal{P}^{\mathrm{wd}}_{\mathcal{M}} \to [0,1]$ be defined as $sol^{\mathcal{M},T}_s[val] := \Pr_{\mathcal{M}[val]}(s \to \Diamond T)$. For a pMDP $\mathcal{M}$, let $minsol^{\mathcal{M},T}_s[val] := \min_{\sigma \in \Sigma} sol^{\mathcal{M}^\sigma,T}_s[val] = \min_{\sigma \in \Sigma} \Pr^\sigma_{\mathcal{M}[val]}(s \to \Diamond T)$. We define $maxsol^{\mathcal{M},T}_s$ analogously as the maximum.*

Let $sol^{\mathcal{M},T}$ denote $sol^{\mathcal{M},T}_{s_\iota}$ with the convention that T is omitted whenever it is clear from the context. On $\mathcal{P}^{\mathrm{gp}}$, $sol^{\mathcal{M}}$ is described by a rational function over the parameters [19, 39], and is computable in $\mathcal{O}\left(poly(|S| \cdot d)^{|X|}\right)$, where d is the maximal degree of polynomials in $\mathcal{M}$'s transitions [33]. The number of resulting monomials is polynomial in $|S|$ and d but exponential in $|X|$. Furthermore, the degree of f and g in the resulting function f/g is upper-bounded by $\ell(d)$ – where ℓ is a linear function.[1] For acyclic pMCs, $sol^{\mathcal{M}}$ is described by a polynomial.

2.2 Existential theory of the reals

Many results in this paper are based on results from the existential theory of the reals [4]. We give a brief recap. We consider the first-order theory of the reals: the set of all valid sentences in the first-order language $(\mathbb{R}, +, \cdot, 0, 1, <)$. The existential theory of the reals restricts the language to (purely) existentially quantified sentences. The complexity of deciding membership, i.e. whether a sentence is (true) in the theory of the reals, is in PSPACE [7] and NP-hard. A careful analysis of its complexity is given in [45]. In particular, deciding membership for sentences with an a-priori fixed upper bound on the number of variables is in polynomial time. ETR denotes the complexity class [48] of problems with a polynomial-time many-one reduction to deciding membership in the existential theory of the reals.

[1] Importantly, this means that if the coefficients and exponents were written in binary for the given pMC then linearly more bits suffice to do the same for the computed rational function.

Table 1 The complexity landscape for reachability in simple pMDPs. All problems are in ETR.

		Fixed # parameters	Arbitrary # parameters	
			well-defined	graph-preserving
pMC	$\exists\text{Reach}^{\geq/\leq}$	in P [33]	— ETR-complete [Thm. 27] —	
	$\exists\text{Reach}^{>}$	”	NP-hard [Thm. 33]	$\exists\text{Reach}^{>}_{\text{wd}}$-complete [Lem. 32]
	$\exists\text{Reach}^{<}$	”	NP-hard [Thm. 33]	$\exists\text{Reach}^{>}_{\text{wd}}$-complete [Lem. 9]
pMDP	$\exists\exists\text{Reach}^{\geq/\leq}$	in NP [Thm. 13]	— ETR-complete — (trivial)	
	$\exists\exists\text{Reach}^{>}$	”	— $\exists\text{Reach}^{>}_{\text{wd}}$-complete [Lem. 34, Cor. 36] —	
	$\exists\exists\text{Reach}^{<}$	”	$\exists\text{Reach}^{<}_{\text{wd}}$-complete [Lem. 34]	$\exists\text{Reach}^{>}_{\text{wd}}$-hard (trivial)
	$\exists\forall\text{Reach}^{\bowtie}$	in NP [Thm. 14]	— ETR-complete [Thm. 37] —	

3 Problem landscape

In this section, we introduce the family of decision problems of our main interest. Let a *simple* pMDP $\mathcal{M}$ with all constants rational, and a set T of target states be the given input. We analyse the decision problems according to whether the set X of parameters from $\mathcal{M}$ has bounded size – with a-priori fixed bound – or arbitrary size.

It remains for us to fix an encoding for rational functions. Henceforth, we assume the coefficients, exponents and constants are all given as binary-encoded integer pairs.

Decision problems. The first problem is the existence of so-called *robust parameter values* or lack thereof. More precisely, the question is whether some instantiation of $\mathcal{M}$ is such that its maximal or minimal probability of eventually reaching T compares with $\frac{1}{2}$ in some desired way. In symbols, for $\mathcal{Q}_1, \mathcal{Q}_2 \in \{\exists, \forall\}$ and $\bowtie \in \{\leq, <, >, \geq\}$, let

$$\mathcal{Q}_1\mathcal{Q}_2\text{Reach}^{\bowtie}_{\text{wd}} \overset{\text{def}}{\iff} \mathcal{Q}_1\ \mathit{val} \in \mathcal{P}^{\text{wd}}, \mathcal{Q}_2\ \sigma \in \Sigma.\ \Pr{}^{\sigma}_{\mathcal{M}[\mathit{val}]}(\lozenge T) \bowtie \frac{1}{2}$$

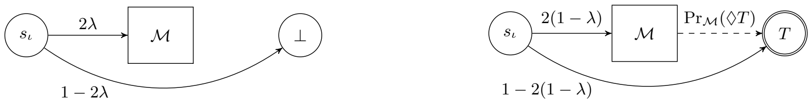
be the problem of interest. We write $\mathcal{Q}_1\mathcal{Q}_2\text{Reach}^{\bowtie}_{\text{gp}}$ whenever $\mathcal{Q}_1$ quantifies over graph-preserving instantiations. We write $\mathcal{Q}_1\mathcal{Q}_2\text{Reach}^{\bowtie}_{*}$ to denote both the wd and gp variants. Furthermore, if $\mathcal{M}$ is a pMC we omit the second quantifier, e.g. $\exists\text{Reach}^{<}_{*}$. Table 1 surveys the results.

▶ **Proposition 7.** *For every $\mathcal{Q}_1, \mathcal{Q}_2 \in \{\exists, \forall\}$ and $\bowtie \in \{\leq, <, >, \geq\}$, $\mathcal{Q}_1\mathcal{Q}_2\text{Reach}^{\bowtie}_{*}$ are decidable in ETR.*

Proof. Both $\exists\forall\text{Reach}^{\bowtie}_{*}$ and $\exists\exists\text{Reach}^{\bowtie}_{*}$ are in ETR (for an encoding, see Appendix B in the technical report). It follows that $\exists\text{Reach}^{\bowtie}_{*}$ are also in ETR. ◀

Problems with fixed threshold. In the above-defined problems, we have fixed a threshold of $\frac{1}{2}$. This is no loss of generality as any *given* rational threshold can be reduced to $\frac{1}{2}$:

▶ Remark 8. An arbitrary threshold $0 < \lambda < 1$, $\lambda \in \mathbb{Q}$, is reducible to $\frac{1}{2}$ by the constructions depicted in Fig. 1: If $\lambda \leq \frac{1}{2}$ then we prepend a transition with probability $p = 2\lambda$ to the initial state and with probability $1 - p$ to a sink state. Otherwise, if $\lambda > \frac{1}{2}$, we prepend a transition with probability $q = 2(1 - \lambda)$ to the initial state and $1 - q$ to the target state. Conversely, the $\frac{1}{2}$ threshold may analogously be reduced to an arbitrary threshold $0 < \lambda < 1$.



(a) $\lambda \leq \frac{1}{2}$ **(b)** $\lambda > \frac{1}{2}$

Figure 1 Reductions to reachability threshold $\lambda = \frac{1}{2}$, cf. Remark 8.

Considerations for the comparison relations.

▶ **Lemma 9.** *For every $\mathcal{Q}_1, \mathcal{Q}_2 \in \{\exists, \forall\}$, there are polynomial-time Karp reductions*

- *among the problems $\mathcal{Q}_1\mathcal{Q}_2\text{Reach}^{>}_{\text{gp}}$ and $\mathcal{Q}_1\mathcal{Q}_2\text{Reach}^{<}_{\text{gp}}$ and*
- *among the problems $\mathcal{Q}_1\mathcal{Q}_2\text{Reach}^{\geq}_{\text{gp}}$ and $\mathcal{Q}_1\mathcal{Q}_2\text{Reach}^{\leq}_{\text{gp}}$.*

The above claim only holds when restricted to graph-preserving parameter spaces.

Semi-continuity. The following theorem formalises an observation in [37, Thm. 5].

▶ **Theorem 10.** *For each simple pMC $\mathcal{M}$, the function sol$^{\mathcal{M}}$ is lower semi-continuous, and continuous on $\mathcal{P}^{\text{gp}}_{\mathcal{M}}$. For acyclic simple pMCs $\mathcal{M}$, sol$^{\mathcal{M}}$ is continuous on $\mathcal{P}^{\text{wd}}_{\mathcal{M}}$.*

Continuity on $\mathcal{P}^{\text{gp}}_{\mathcal{M}}$ follows as *sol*$^{\mathcal{M}}$ is a rational function bounded by $[0,1]$ on all well-defined points [44]. Graph non-preserving instantiations might yield additional sink states in the induced MC, therefore, the probability may drop when changing a parameter instantiation, e.g. $p = 0$ with an single outgoing transition with probability p and a self-loop with probability $1-p$.The semi-continuity is the main reason that we do not have symmetric entries for upper and lower bounds in Table 1.

The following result follows immediately from properties of (semi-)continuous functions.

▶ **Corollary 11.** *For all pMDPs, the functions minsol$^{\mathcal{M}}$ and maxsol$^{\mathcal{M}}$ are lower semi-continuous and have a minimum. For acyclic pMDPs, these functions are continuous.*

▶ **Corollary 12.** *For acyclic pMCs, sol$^{\mathcal{M}}$ is described by a polynomial even on $\mathcal{P}^{\text{wd}}_{\mathcal{M}}$.*

4 Fixing the number of parameters

In this section, we assume that the number of parameters is fixed. We focus ourselves on graph-preserving instantiations, as the analysis of pMDP $\mathcal{M}$ and $\mathcal{P}^{\text{wd}}_{\mathcal{M}}$ corresponds to analysing constantly many pMDPs $\mathcal{M}'$ on $\mathcal{P}^{\text{gp}}_{\mathcal{M}'}$, cf. Rem. 3.

Upper bounds. Below, we establish NP membership for all variants.

▶ **Lemma 13.** *In the fixed parameter case, $\exists\exists\text{Reach}^{\bowtie}_{*}$ is in NP.*

Proof. Guess a memoryless scheduler. Construct the induced pMC, and verify it in P. ◀

▶ **Theorem 14.** *In the fixed parameter case, $\exists\forall\text{Reach}^{\bowtie}_{*}$ is in NP.*

In the non-parametric case, a scheduler σ of an MDP is called *minimal* if it minimises $\Pr^{\sigma}(\Diamond T)$, i.e. if $\sigma \in \text{argmin}_{\sigma' \in \Sigma} \Pr^{\sigma'}(\Diamond T)$. Consider the probabilities $x_s = \Pr^{\sigma}(s \to \Diamond T)$ for $s \in S$. It is well-known (see, e.g., [43]) that σ is minimal if and only if $x_s \leq \sum_{s' \in S} P(s, a, s') \cdot x_{s'}$ holds for all $s \in S$ and $a \in \mathit{Act}(s)$. (There is a similar condition for *maximal* schedulers.)

The minimality criterion can be lifted to the parametric case: Suppose $R \subseteq \mathcal{P}^{\mathrm{wd}}$ is a graph-consistent region and let $f_s = sol_s^{\mathcal{M}^\sigma}$. Then σ is *somewhere* minimal on R if and only if there exists some $val \in R$ such that

$$f_s[val] \leq \sum_{s' \in S} P(s, a, s') \cdot f_{s'}[val] \tag{1}$$

for all $s \in S$ and $a \in Act$. (For *everywhere* minimal strategies, a universal quantification over val yields the correct criterion).

▶ **Lemma 15.** *In the fixed parameter case, checking whether a given strategy is somewhere (resp. everywhere) minimal (resp. maximal) on $\mathcal{P}^{\mathrm{wd}}$ is in P.*

Proof sketch. Condition (1) can be reformulated as the ETR formula with $|X|$ many variables

$$\Psi = \exists val \colon \Phi_R(val) \longrightarrow \Phi_\sigma(val) \tag{2}$$

where $\Phi_R(val)$ is a formula which is true if and only if $val \in R$ and

$$\Phi_\sigma(val) = \bigwedge_{s \in S} \bigwedge_{a \in Act} \left(g_s[val] \cdot \prod_{s' \neq s} h_{s'}[val] \;\leq\; \sum_{s' \in S} P(s, a, s') \cdot g_{s'}[val] \cdot \prod_{s'' \neq s'} h_{s''}[val] \right) \tag{3}$$

where ${}^{g_s}/_{h_s} = f_s$ for $g_s, h_s \in \mathbb{Q}[val]$. (W.l.o.g. it holds that $h_s[val] > 0$ for all $val \in R$.) ◀

Proof sketch of Thm. 14. Consider $\bowtie = \geq$: Guess a somewhere minimal scheduler. Check its minimality similar to Lem. 15, but extended to simultaneously ensure that the induced pMC satisfies the threshold. The other relations in $\bowtie$ are analogous. ◀

Sets of optimal schedulers. For the problems $\forall\forall\mathrm{Reach}_*^{\bowtie}$ and $\forall\exists\mathrm{Reach}_*^{\bowtie}$ (with fixed parameters) we already have coNP-membership (as we considered their complements before). It is tempting to assume that their NP-membership can be established analogous to above, relying on *everywhere* optimal schedulers which, according to Lem. 15, can also be verified in polynomial time. However, such schedulers do not necessarily exist. What we need instead is a *set* of somewhere optimal schedulers covering the entire parameter space – a so called *optimal-scheduler set (OSS)*.

▶ **Definition 16** (Optimal scheduler set). *A set $\Omega \subseteq \Sigma$ is called an* optimal scheduler set *(OSS) on $R \subseteq \mathcal{P}^{\mathrm{wd}}$ if*

$$\forall val \in R, \exists \sigma \in \Omega.\ \Pr^{\sigma}_{\mathcal{M}[val]}(\lozenge T) = \max_{\sigma' \in \Sigma} \Pr^{\sigma'}_{\mathcal{M}[val]}(\lozenge T),$$

i.e. Ω contains a maximal scheduler for every point in the region R. The notion can be analogously defined for minimal schedulers.

An OSS of minimal cardinality is called a *minimal optimal scheduler set* (MOSS). For many applications it is appropriate to describe a region R via a quantifier-free ETR-formula Φ_R with $|X|$ free variables such that $Sat(\Phi_R) = R$. In that case, we have the following:

▶ **Theorem 17.** *In the fixed parameter case, checking whether a given $\Omega \subseteq \Sigma$ constitutes an OSS on $R = Sat(\Phi_R)$ can be done in time polynomial in the size of $\mathcal{M}$, Ω and Φ_R.*

Proof. For every $\sigma \in \Omega$, we construct the formulas Φ_σ as in (3) in polynomial time. Then, we check whether the fixed-parameter ETR-formula

$$\exists val \colon \Phi_R(val) \longrightarrow \bigwedge_{\sigma \in \Omega} \neg\Phi_\sigma(val)$$

is unsatisfiable (also in polynomial time). If yes, return true and otherwise false. ◀

Checking whether a set is a MOSS then additionally requires consideration of all the subsets with one element removed (again in polynomial time).

▶ **Lemma 18.** *If the size of a MOSS on $\mathcal{P}^{\mathrm{wd}}$ is polynomially bounded for fixed-parameter pMDPs, then $\exists\exists\mathrm{Reach}_*^{\bowtie}$ and $\exists\forall\mathrm{Reach}_*^{\bowtie}$ are in coNP.*

The proof considers the complement of $\exists\forall\mathrm{Reach}_*^{\bowtie}$, that is $\forall\exists\mathrm{Reach}_*^{\bar{\bowtie}}$. Under the assumption in the lemma, it now suffices to guess a MOSS and verify $\exists\mathrm{Reach}_*^{\bar{\bowtie}}$ on the induced pMCs in polynomial time, showing that the complement is in NP.
In the arbitrary parameter case, we obtain an exponential lower bound on the MOSS size:

▶ **Lemma 19.** *There exists a family $(\mathcal{M}_n)_{n\in\mathbb{N}}$ of simple pMDPs with $n+2$ states s.t. $|\Omega| \geq 2^n$ for any OSS Ω on $\mathcal{P}^{\mathrm{wd}}$, i.e., the size of a MOSS can grow exponentially in the pMDP's size.*

This lemma, and what follows below, consider the unbounded parameter case, i.e., from now on, parameters are part of the input.

5 The expressiveness of simple pMCs

We investigate the relation between polynomial inequalities and the $\exists$Reach problems. The first lemma in this section is a key ingredient for our complexity analysis later on.

▶ **Lemma 20** (Chonev's trick [16, Remark 7]). *Let $f \in \mathbb{Q}[X]$ be a polynomial, $\mu \in \mathbb{Q}$ and $0 < \lambda < 1$. There exists a simple acyclic pMC $\mathcal{M}$ with a target state T such that for all $val\colon X \to [0,1]$ and all comparison relations $\bowtie \in \{<, \leq, \geq, >, =\}$ it holds that*

$$f[val] \bowtie \mu \iff \mathrm{Pr}_{\mathcal{M}[val]}(\lozenge T) \bowtie \lambda.$$

Moreover, if d is the total degree of f, t the number of terms in f and κ a bound on the (bit-)size of the coefficients and the thresholds μ, λ, then $\mathcal{M}$ can be constructed in time $\mathcal{O}(poly(d, t, \kappa))$.

▶ **Example 21.** Consider the inequality $-2x^2y + y \geqslant 5$. We reformulate this to: $2 \cdot ((1-x)xy + (1-x)y + (1-y) - 1) + y \geqslant 5$ and then to $2{\cdot}(1-x)xy + 2{\cdot}(1-x)y + 2{\cdot}(1-y) + y \geqslant 7$. Observe that both sides now only contain positive coefficients. Furthermore, observe that we wrote the left-hand side as sum of products over $\{x, 1-x, y, 1-y\}$. After rescaling (with $\frac{1}{8}$), we can construct the pMC $\mathcal{M}$ depicted in Fig. 2a and set $\lambda = \frac{7}{8}$.

Checking a bound on a given polynomial over X thus is equivalent to checking a bound on a reachability probability in a simple acyclic pMC over X. For the fixed parameter case, this gives rise to the following equivalence relating arbitrary pMCs to simple acyclic pMCs.

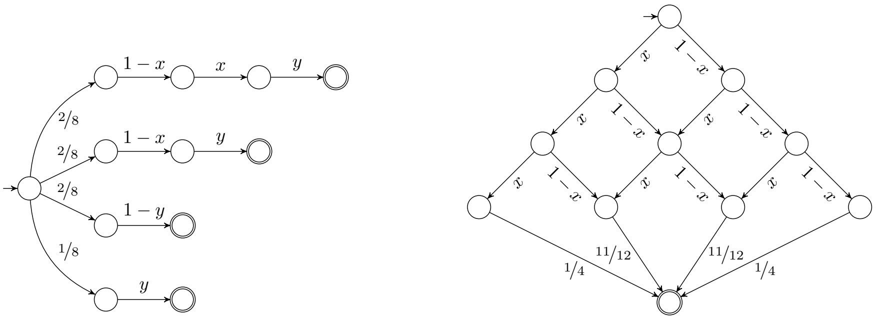
▶ **Theorem 22.** *For any non-simple pMC $\mathcal{M}$ with $\mathcal{P}_{\mathcal{M}}^{\mathrm{gp}} = (0,1)^X$ there exists a simple acyclic pMC $\mathcal{M}'$ such that*

$$\{val \in \mathcal{P}_{\mathcal{M}}^{\mathrm{gp}} \mid \mathrm{Pr}_{\mathcal{M}[val]}(\lozenge T) \bowtie \lambda\} = \{val \in \mathcal{P}_{\mathcal{M}'}^{\mathrm{gp}} \mid \mathrm{Pr}_{\mathcal{M}'[val]}(\lozenge T) \bowtie \lambda\}.$$

In the fixed parameter case, $\mathcal{M}'$ can be computed in polynomial time.

The proof is constructive: one first computes the (rational function) $sol_{\mathcal{M}}$, reformulates that as a polynomial constraint, and casts that into a simple acyclic pMC using Lemma 20.

The goal of the rest of this section is to prove a result which is, in a sense, a stronger version of Lemma 20. In particular, we want to describe polynomials by $sol^{\mathcal{M}}$ for an acyclic pMC. We call a polynomial $f \in \mathbb{Q}[X]$ *adequate* if $0 < f[val] < 1$ for all $val\colon X \to (0,1)$ and $0 \leq f[val] \leq 1$ for all $val\colon X \to [0,1]$. Note that *$sol^{\mathcal{M}}$ is an adequate polynomial if $\mathcal{M}$ is both simple and acyclic*, and there is no acyclic pMC $\mathcal{M}$ (with a single parameter) such that $sol_{\mathcal{M}}$ is *not* adequate – except where $sol_{\mathcal{M}} = 0$ or $sol_{\mathcal{M}} = 1$.



(a) The reachability probability in $\mathcal{M}[val]$ is at least $\frac{7}{8}$ iff $(-2x^2y + y)[val] \geq 5$.

(b) The reachability probability from source to target equals $f = 2x \cdot (1-x) + \frac{1}{4}$.

Figure 2 Examples for the strong connection between polynomial (inequalities) and pMCs. Transitions to the sink are not depicted for conciseness.

▶ **Theorem 23.** *Let $f \in \mathbb{Q}[x]$ be a (univariate) adequate polynomial. There exists a simple acyclic pMC $\mathcal{M}$ with a target state T such that $f = sol^{\mathcal{M}}$.*

Our construction of a pMC for some adequate polynomial is based on the following result:

▶ **Lemma 24** (Handelman's theorem [30])**.** *Let $\beta_1 \geq 0, \dots, \beta_\ell \geq 0$ be linear constraints that define a compact convex polyhedron $P \subseteq \mathbb{R}^n$ with interior. If a polynomial $f \in \mathbb{R}[x_1, \dots, x_n]$ is strictly positive on P, then f may be written as*

$$f = \sum_{i=1}^{k} \lambda_i h_i \tag{4}$$

where $h_i = \beta_1^{e_{i,1}} \cdot \ldots \cdot \beta_\ell^{e_{i,\ell}}$ for some natural exponents $e_{i,j}$ and real coefficients $\lambda_i > 0$.

Form (4) is called a *Handelman representation* of f w.r.t. $\beta_1, \dots, \beta_\ell$. The next lemma states the existence of a specific Handelman representation which we can map to a pMC.

▶ **Lemma 25.** *Let $f \in \mathbb{Q}[x]$ be an adequate polynomial. There exists an $n \geq 0$ such that*

$$f = \sum_{k=0}^{n} p_k \cdot \binom{n}{k} \cdot x^{n-k} \cdot (1-x)^k \qquad \text{with } p_k \in [0,1] \text{ for all } 0 \leq k \leq n \tag{5}$$

▶ **Example 26.** Consider $f = 2x \cdot (1-x) + \frac{1}{4}$ which is strictly positive on $[0,1]$ and already in a Handelman representation. Following the proof of Lem. 25, we find that

$$f = \frac{1}{4}\binom{0}{3}x^3 + \frac{11}{12}\binom{1}{3}x^2(1-x) + \frac{11}{12}\binom{2}{3}x(1-x)^2 + \frac{1}{4}\binom{3}{3}(1-x)^3$$

The construction (described in the proof of Thm. 23) yields the pMC depicted in Fig. 2b.

6 The complexity of reachability in pMCs

We improve lower bounds for ∃Reach problems. The results depend on the comparison type:

Nonstrict inequalities. This paragraph is devoted to proving the following theorem:

▶ **Theorem 27.** $\exists\mathrm{Reach}_*^{\leq}$, $\exists\mathrm{Reach}_*^{\geq}$ *are all ETR-complete (even for acyclic pMCs).*

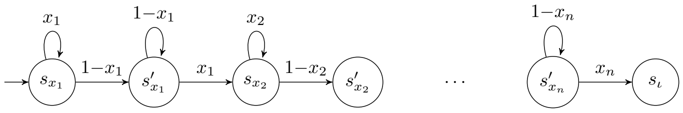


Figure 3 Gadget for the proof of Lemma 32.

▶ **Definition 28.** *The decision problem* modified-closed-bounded-4-feasibility *(mb4FEAS-c) asks: Given a (non-negative) quadric polynomial f, $\exists val\colon X \to [0,1]$ s.t. $f[val] \leq 0$? The* modified-open-bounded-4-feasibility *(mb4FEAS-o) is analogously defined with val ranging over* $(0,1)$.

This problem easily reduces to its $\geq$-variant by multiplying f with -1.

▶ **Lemma 29.** *The problems mb4FEAS-c and mb4FEAS-o are ETR-hard.*

Essentially, one reduces from the existence of common roots of quadratic polynomials lying in a unit ball, which is known to be ETR-complete [47, Lemma 3.9]. The reduction to mb4FEAS follows the reduction[2] between unconstrained variants (i.e., variants in which the position of the root is not constrained) of the same decision problems [48, Lemma 3.2].

▶ Remark 30. Observe that there may be exactly one satisfying assignment to mb4FEAS-o/c, which may be irrational. In contrast, if there exists a satisfying assignment for $f > 0$, then there exist infinitely many satisfying (rational) assignments. To the best of our knowledge, the complexity of a variant of mb4FEAS-o/c with strict bounds is open. Therefore, we have no ETR-hardness for $\exists$Reach with strict bounds. In general, *conjunctions* of strict inequalities are also ETR-complete [48]. We exploit this in the proof of Thm. 37 on page 12.

Proof of Thm. 27. The reduction from mb4FEAS-c to $\exists\text{Reach}^{\leq}_{\text{wd}}$ is a straightforward application of Lemma 20 with $\mu = 0$ and $\lambda = \frac{1}{2}$. For $\exists\text{Reach}^{\leq}_{\text{gp}}$, we reduce from the open variant and notice that as the construction in Lemma 20 preserves all satisfying instantiations $val\colon X \to [0,1]$ it, in particular, also preserves them on the graph-preserving parameter space. For $\geq$, we apply Lemma 20 on $-f$. ◀

The tight complexity class shows that the assumption of simplicity is not a real restriction. Furthermore, a similar construction can be used for (sufficiently large[3], linear) subsets of the parameter space. In particular, methods [18, 44] targeted at a variant of $\exists$Reach considering a so-called ϵ-preserving parameter space ($[\epsilon,\, 1-\epsilon]^k$) target an ETR-complete problem.

Strict inequalities. In this paragraph, the main result is:

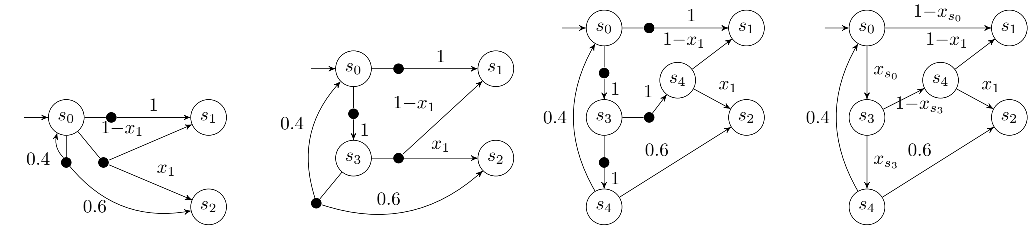
▶ **Theorem 31.** $\exists\text{Reach}^{>}_{*}$ *and* $\exists\text{Reach}^{<}_{*}$ *are NP-hard.*

The gadget in Fig. 3 ensures that for any graph non-preserving instantiation, the probability to reach the target is 0, while it does not affect reachability probabilities for graph-preserving instantiations. Together with semi-continuity of the solution function, we deduce that assuming graph-preservation is equivalent to not making this assumption:

▶ **Lemma 32.** *There are polynomial-time Karp reductions among* $\exists\text{Reach}^{>}_{gp}$ *and* $\exists\text{Reach}^{>}_{\text{wd}}$.

[2] Essentially the polynomial f in mb4FEAS is constructed by taking the sum-of-squares of the quadratic polynomials, and further operations are adequatly shifting the polynomial.

[3] The bounds should be at least δ apart, where δ requires at most single-exponentially many bits.



(a) simple pMDP. **(b)** intermediate step. **(c)** simple bin.-dec. **(d)** simple pMC.

Figure 4 From simple pMDP to simple pMC.

We may thus turn our attention to well-defined parameter spaces: The decision problem

$$\exists\text{Reach}^{\geq 1}_{\text{wd}} \overset{\text{def}}{\iff} \exists val \in \mathcal{P}^{\text{wd}}_{\mathcal{M}}.\ \Pr_{\mathcal{M}[val]}(\lozenge T) \geq 1$$

is NP-complete [16, Thm. 3]. A more refined analysis of the 3SAT-reduction yields:

▶ **Theorem 33.** *$\exists\text{Reach}^{>}_{\text{wd}}$ and $\exists\text{Reach}^{<}_{\text{wd}}$ are NP-hard.*

Proof of Thm. 31. Lem. 32, Thm. 33, and Lem. 9 together imply Thm. 31. ◀

This concludes our complexity analysis for pMCs.

7 The complexity of reachability in pMDPs

7.1 Exists-exists reachability

By definition, every pMC is a pMDP. Conversely, from any pMDP we can construct a pMC such that their $\exists\exists\text{Reach}^{\bowtie}_{\text{wd}}$ problems coincide. A similar construction relates pMCs to the existence of optimal randomised memoryless strategies in partially observable MDPs [37].

▶ **Lemma 34.** *There are polynomial-time Karp reductions among $\exists\text{Reach}^{\bowtie}_{\text{wd}}$ and $\exists\exists\text{Reach}^{\bowtie}_{\text{wd}}$.*

We outline the steps in Fig. 4 and in the description below.

Binary-decision pMDPs. The first step of the translation consists in restricting the non-determinism resolved by a scheduler to (at most) two options from every state. A binary-decision pMDP is a pMDP such that $|Act(s)| \leq 2$ for all states $s \in S$ and if $|Act(s)| = 2$ then $\forall a \in Act(s), \forall s' \in S,\ P(s,a,s') \in \{0,1\}$. Any pMDP can be transformed (in polynomial time) into a binary-decision pMDP by introducing auxiliary states and simulating k-ary non-deterministic choice using a binary-tree-like scheme in which all non-Dirac transitions are pushed to the leaves (see, e.g., [37,44,49]). Such a construction preserves simplicity.

From non-determinism to parameters. For a given binary-decision pMDP $\mathcal{M}$, we may replace all non-determinism by parameters, inspired by [28,37]. We introduce fresh variables $X_S = \{x_s \mid s \in S\}$. In $\mathcal{M}$, for any state s with $Act(s) = \{a, a'\}$ we replace

- the unique transition $P(s,a,s') = 1$ by $P(s,a,s') = x_s$
- the unique transition $P(s,a',s') = 1$ by $P(s,a',s') = 1 - x_s$.

The outcome is a simple pMC $\mathcal{M}'$. To translate instantiations into schedulers, and vice versa, it is helpful to consider randomised schedulers. Observe that, by Rem. 5, instantiations which translate into such schedulers are always dominated by deterministic ones.
Using the previously described construction, we obtain the following.

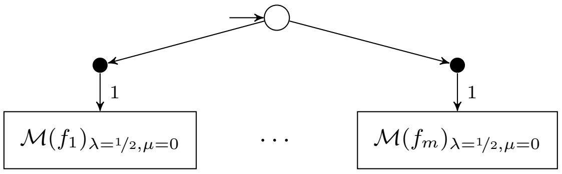

1
1
$\mathcal{M}(f_1)_{\lambda=1/2,\mu=0}$
...
$\mathcal{M}(f_m)_{\lambda=1/2,\mu=0}$

Figure 5 Construction for the proof of Thm. 37.

▶ **Lemma 35.** *For all simple pMDPs $\mathcal{M}$ one can construct in polynomial time a (linearly larger) simple pMC $\mathcal{M}'$ s.t.*

$$\left(\exists val \in \mathcal{P}^{\mathrm{wd}}_{\mathcal{M}}, \exists \sigma \in \Sigma.\ \Pr^{\sigma}_{\mathcal{M}[val]}(\Diamond T) \bowtie \frac{1}{2}\right) \iff \left(\exists val \in \mathcal{P}^{\mathrm{wd}}_{\mathcal{M}'}.\ \Pr_{\mathcal{M}'[val]}(\Diamond T) \bowtie \frac{1}{2}\right).$$

▶ **Corollary 36.** *There are polynomial-time Karp reductions among* $\exists\exists\mathrm{Reach}^{>}_{\mathrm{gp}}$ *and* $\exists\mathrm{Reach}^{>}_{\mathrm{wd}}$.

Proof. Minor adaptions in the proofs of Lemma 34 and Lemma 32. ◀

7.2 Exists-forall reachability

Contrary to pMCs, we obtain ETR-completeness in pMDPs for any comparison relation:

▶ **Theorem 37.** $\exists\forall\mathrm{Reach}^{\bowtie}_{*}$ *are all ETR-complete (even for acyclic pMDPs with a single non-deterministic state).*

For the strict relations, we use a different problem to reduce from.

▶ **Definition 38.** *The decision problem* bounded-conjunction-of-inequalities *(bcon4INEQ-c) asks: Given a family of quadric polynomials $f_1, \ldots, f_m$, $\exists val : X \to [0,1]$ s.t. $\bigwedge_{i=1}^{m} f_i[val] < 0$? The open variant (bcon4INEQ-o) can be defined analogously.*

By a reduction from mb4FEAS (adapted from [48, Thm 4.1]):

▶ **Lemma 39.** *The bcon4INEQ-o/c problems are ETR-hard.*

Proof sketch of Thm. 37. ETR-hardness for non-strict inequalities follows from Thm. 27. For strict inequalities, we reduce from bcon4INEQ-o/c: Generalise the construction from Thm. 27: Build a pMDP as in Fig. 5 with pMCs $\mathcal{M}(f_i)_{\lambda=1/2,\mu=0}$ created by Lemma 20. ◀

Relation to stochastic games

We argue that pMDPs are – in a sense – a generalisation of *Concurrent Stochastic Reachability Games* (CSRG), a model which has been extensively studied [11,12,20,31,52].

Playing a stochastic game. A CSRG is a two-player game $\mathcal{G}$ played on a finite set S of states. The objective of player I is to reach a target states $T \subseteq S$ while player II has to avoid ever reaching a state in T. A play of $\mathcal{G}$ begins in an initial state s_ι and proceeds as follows: In state s, both players I and II *concurrently* select an action $a \in A_s$ (resp. $b \in B_s$), the finite set of actions available to player I (resp. II) in state s. The game then picks a successor state s' according to a fixed probability distribution $P(\cdot|s,a,b)$ over S, and the play continues in s'. The transition from s to s' is called a *round* of $\mathcal{G}$. Player I wins $\mathcal{G}$ once a state in T is reached. Otherwise, if a target is never reached, then II wins.

A *strategy* σ of a player is, in essence, a scheduler. However, strategies in a CSRG map state-action sequences $s_0(a_0, b_0) \dots s_{k-1}(a_{k-1}, b_{k-1})s_k$ to a probability distribution over the actions A_{s_k} (resp. B_{s_k}) available in the current state s_k. We call σ a *stationary* strategy if it does not depend on the history but only on the current state, i.e. it is a randomised memoryless scheduler. Let Σ^i denote the set of stationary strategies for player $i \in \{\mathrm{I}, \mathrm{II}\}$.

Instantiations and MDPs. The *instantiation* $\mathcal{G}^\sigma$ of $\mathcal{G}$ with a stationary strategy for player I is the structure obtained by forcing player I to follow σ. Notice that $\mathcal{G}^\sigma$ is a finite MDP $\mathcal{M}$. Its transition probability function $P_\mathcal{M}$ is obtained by letting

$$P_\mathcal{M}(s, b, s') = \sum_{a \in A_s} \sigma(a|s) P(s'|s, a, b) \tag{6}$$

for all $s, s' \in S$ and actions $b \in B_s$ of player II. (Instantiations are defined completely symmetrically for strategies of player II.) Conversely, every MDP may be viewed as a CSRG where $|A_s| = 1$ (or $|B_s| = 1$) for all $s \in S$, i.e. one of the players does never have any choice.

Value of a CSRG. Let $\mathrm{Pr}^{\sigma,\tau}_\mathcal{G}(\Diamond T)$ be the probability that T is reached if player I plays according to σ and player II according to τ. The *value* of $\mathcal{G}$ is defined as $V(\mathcal{G}) \coloneqq \sup_\sigma \inf_\tau \mathrm{Pr}^{\sigma,\tau}_\mathcal{G}(\Diamond T)$ where the sup and inf range over all strategies of both players respectively. Intuitively, it is the maximal winning probability of player I that can be guaranteed against all strategies of player II. The existence of stationary optimal strategies for player II [32, 41] allows us to encode a CSRG in a pMDP by replacing the universal player II with parameters:

▶ **Theorem 40.** *For any given CSRG $\mathcal{G}$, there exists a simple pMDP $\mathcal{M}$ such that*

$$V(\mathcal{G}) = \min_{\tau \in \Sigma^{\mathrm{II}}} \max_{\sigma \in \Sigma^{\mathrm{I}}} \mathrm{Pr}^{\sigma,\tau}_\mathcal{G}(\Diamond T) = \min_{val \in \mathcal{P}^{\mathrm{wd}}} \max_{\sigma \in \Sigma} \mathrm{Pr}^{\sigma}_{\mathcal{M}[val]}(\Diamond T)$$

and $\mathcal{M}$ can be computed in polynomial time (in the size of $\mathcal{G}$).

As a direct consequence, we obtain CSRG-hardness.

▶ **Corollary 41.** *Determining whether $V(\mathcal{G}) \trianglelefteq \lambda$, for $\lambda \in \mathbb{Q}$, reduces to $\exists\forall\mathrm{Reach}^{\trianglelefteq}_{\mathrm{wd}}$.*

It follows from [11, Thms. 6 and 12] that optimal rational instantiations may be complex.

▶ **Theorem 42.** *There are pMDPs for which rational optimal and ε-optimal parameter values minimising the value $\max_{\sigma \in \Sigma} \mathrm{Pr}^{\sigma}_{\mathcal{M}[val]}(\Diamond T)$ require exponentially-many bits to be written as a binary-encoded integer-pair.*

8 Robust reachability

In this section, we briefly consider pMDPs in which we focus on obtaining (robust) schedulers rather than (robust) parameter values: We swap the quantification order from the $\mathcal{Q}_1\mathcal{Q}_2\mathrm{Reach}$ problem. Intuitively, we ask whether some scheduler gives guarantees on the maximal or minimal probability of all instantiations of the pMDP eventually reaching T. Formally, for each $\mathcal{Q}_1, \mathcal{Q}_2 \in \{\exists, \forall\}$ and $\bowtie \in \{\leq, <, >, \geq\}$, let

$$\mathcal{Q}_1\mathcal{Q}_2\mathrm{RobReach}^{\bowtie}_{\mathrm{wd}} \overset{\mathrm{def}}{\iff} \mathcal{Q}_1 \sigma \in \Sigma,\ \mathcal{Q}_2 val \in \mathcal{P}^{\mathrm{wd}}.\ \mathrm{Pr}^{\sigma}_{\mathcal{M}[val]}(\Diamond T) \bowtie \frac{1}{2}.$$

We adopt the same conventions as for the $\mathcal{Q}_1\mathcal{Q}_2\mathrm{Reach}$ problem when considering graph-preserving instantiations. Variants which use the same quantifier twice, or consider pMCs yield the same results as for $\mathrm{Reach}^{\bowtie}$, and are therefore omitted.

Robust strategies have been widely studied in the field of operations research (see, e.g., [40, 57]) and are the main focus of *reinforcement learning* [55]. It is known that the robust-reachability problem as defined above is not the most general question one can ask. Indeed, we restrict our attention to *memoryless schedulers* while, in general, optimal robust schedulers require memory and randomisation [1].

Our interest in the robust-reachability problem is twofold. First, it naturally corresponds to the quantifier-swapped version of the reachability problem. Second, memoryless schedulers are desirable in practice for their comprehensibility and ease of implementation.

▶ **Theorem 43.** *In the fixed parameter case,* $\exists\forall\mathrm{RobReach}_*^{</>}$ *are NP-complete. NP-hardness holds even for acyclic pMDPs with a single parameter.*

Proof sketch. Membership in NP is analogous to Lem. 13. NP-hardness is based on a reduction from 3-SAT, with a construction similar to Fig. 5. ◀

▶ **Proposition 44.** *The decision problems* $\exists\forall\mathrm{RobReach}_*^{\bowtie}$ *are NP-hard and coNP-hard, and in PSPACE. For non-strict inequalities, the problems are coETR-hard.*

Proof. NP-hardness follows from Thm. 43, coNP/coETR-hardness follows from Thm. 31, Thm. 33, and Thm. 27, respectively. Iterating over all (finitely many) schedulers, check each scheduler in ETR or in coETR (and thus in PSPACE). ◀

Consequently, it is unlikely that either of the problems are in ETR or coETR, as then ETR and coETR would coincide (which is not impossible, but unlikely [48]).

9 Conclusions

We have studied the complexity of various reachability problems for *simple* pMCs and pMDPs. All the problems we have considered are easily seen to be solvable in PSPACE via reductions to the existential theory of the reals. We have complemented this observation with lower bounds, i.e. ETR hardness for several versions of the problem both for (tree-like) pMCs and pMDPs. These lower bounds naturally extend to general pMCs and pMDPs, and to expected reward measures.

We have given an NP decision procedure for pMDPs with a fixed number of parameters. The exact complexity of pMDP reachability problems with this restriction remains open, and our upper bounds do not straightforwardly generalise beyond simple pMDPs (see Rem. 3).

Finally, we have established a tight connection between polynomials and pMCs (even beyond [16]). However, our results do not allow us to conclude whether there always are "small" pMCs for every polynomial. Such a result would provide more evidence of ETR being the right framework to solve problems for our parametric models.

References

1 Sebastian Arming, Ezio Bartocci, and Ana Sokolova. SEA-PARAM: exploring schedulers in parametric MDPs. In *QAPL@ETAPS*, volume 250 of *EPTCS*, pages 25–38, 2017.

2 Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT Press, 2008.

3 Ezio Bartocci, Radu Grosu, Panagiotis Katsaros, C. R. Ramakrishnan, and Scott A. Smolka. Model Repair for Probabilistic Systems. In *TACAS*, volume 6605 of *LNCS*, pages 326–340. Springer, 2011.

4 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. Existential theory of the reals. *Algorithms in Real Algebraic Geometry*, pages 505–532, 2006.

5 Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The Complexity of Decentralized Control of Markov Decision Processes. *Math. Oper. Res.*, 27(4):819–840, 2002.

6 Luca Bortolussi and Simone Silvetti. Bayesian Statistical Parameter Synthesis for Linear Temporal Properties of Stochastic Models. In *TACAS (2)*, volume 10806 of *LNCS*, pages 396–413. Springer, 2018.

7 John F. Canny. Some Algebraic and Geometric Computations in PSPACE. In *STOC*, pages 460–467. ACM, 1988.

8 Milan Ceska, Frits Dannenberg, Nicola Paoletti, Marta Kwiatkowska, and Lubos Brim. Precise parameter synthesis for stochastic biochemical systems. *Acta Inf.*, 54(6):589–623, 2017.

9 Krishnendu Chatterjee. Robustness of Structurally Equivalent Concurrent Parity Games. In *FOSSACS*, volume 7213 of *LNCS*, pages 270–285. Springer, 2012.

10 Krishnendu Chatterjee, Martin Chmelik, and Jessica Davies. A Symbolic SAT-Based Algorithm for Almost-Sure Reachability with Small Strategies in POMDPs. In *AAAI*, pages 3225–3232. AAAI Press, 2016.

11 Krishnendu Chatterjee, Kristoffer Arnsfelt Hansen, and Rasmus Ibsen-Jensen. Strategy Complexity of Concurrent Safety Games. In *MFCS*, volume 83 of *LIPIcs*, pages 55:1–55:13. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

12 Krishnendu Chatterjee and Thomas A. Henzinger. A survey of stochastic omega-regular games. *J. Comput. Syst. Sci.*, 78(2):394–413, 2012.

13 Taolue Chen, Yuan Feng, David S. Rosenblum, and Guoxin Su. Perturbation Analysis in Verification of Discrete-Time Markov Chains. In *CONCUR*, volume 8704 of *LNCS*, pages 218–233. Springer, 2014.

14 Taolue Chen, Ernst Moritz Hahn, Tingting Han, Marta Z. Kwiatkowska, Hongyang Qu, and Lijun Zhang. Model Repair for Markov Decision Processes. In *TASE*, pages 85–92. IEEE Computer Society, 2013.

15 Taolue Chen, Tingting Han, and Marta Z. Kwiatkowska. On the complexity of model checking interval-valued discrete time Markov chains. *Inf. Process. Lett.*, 113(7):210–216, 2013.

16 Ventsislav Chonev. Reachability in Augmented Interval Markov Chains. *CoRR*, abs/1701.02996, 2017. `arXiv:1701.02996`.

17 Anne Condon. *Computational models of games.* ACM distinguished dissertations. MIT Press, 1989.

18 Murat Cubuktepe, Nils Jansen, Sebastian Junges, Joost-Pieter Katoen, and Ufuk Topcu. Synthesis in pMDPs: A Tale of 1001 Parameters. In *ATVA*, volume 11138 of *LNCS*, pages 160–176. Springer, 2018.

19 Conrado Daws. Symbolic and Parametric Model Checking of Discrete-Time Markov Chains. In *ICTAC*, volume 3407 of *LNCS*, pages 280–294. Springer, 2004.

20 Luca de Alfaro, Thomas A. Henzinger, and Orna Kupferman. Concurrent reachability games. *Theor. Comput. Sci.*, 386(3):188–217, 2007.

21 Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius, Matthias Volk, Harold Bruintjes, Joost-Pieter Katoen, and Erika Ábrahám. PROPhESY: A PRObabilistic ParamEter SYnthesis Tool. In *CAV (1)*, volume 9206 of *LNCS*, pages 214–231. Springer, 2015.

22 Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A Storm is Coming: A Modern Probabilistic Model Checker. In *CAV (2)*, volume 10427 of *LNCS*, pages 592–600. Springer, 2017.

23 Karina Valdivia Delgado, Scott Sanner, and Leliane Nunes de Barros. Efficient solutions to factored MDPs with imprecise transition probabilities. *Artif. Intell.*, 175(9-10):1498–1527, 2011.

24 Antonio Filieri, Giordano Tamburrelli, and Carlo Ghezzi. Supporting Self-Adaptation via Quantitative Verification and Sensitivity Analysis at Run Time. *IEEE Trans. Software Eng.*, 42(1):75–99, 2016.

25 Paul Gainer, Ernst Moritz Hahn, and Sven Schewe. Accelerated Model Checking of Parametric Markov Chains. In *ATVA*, volume 11138 of *LNCS*, pages 300–316. Springer, 2018.

26 Sergio Giro, Pedro R. D'Argenio, and Luis María Ferrer Fioriti. Distributed probabilistic input/output automata: Expressiveness, (un)decidability and algorithms. *Theor. Comput. Sci.*, 538:84–102, 2014.

27 Robert Givan, Sonia Leach, and Thomas Dean. Bounded-parameter Markov decision processes. *Artif. Intell.*, 122(1-2):71–109, 2000.

28 Ernst Moritz Hahn, Tingting Han, and Lijun Zhang. Synthesis for PCTL in Parametric Markov Decision Processes. In *NASA Formal Methods*, volume 6617 of *LNCS*, pages 146–161. Springer, 2011.

29 Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic reachability for parametric Markov models. *STTT*, 13(1):3–19, 2010.

30 David Handelman. Representing polynomials by positive linear functions on compact convex polyhedra. *Pacific Journal of Mathematics*, 132(1):35–62, 1988.

31 Kristoffer Arnsfelt Hansen, Michal Koucký, and Peter Bro Miltersen. Winning Concurrent Reachability Games Requires Doubly-Exponential Patience. In *LICS*, pages 332–341. IEEE Computer Society, 2009.

32 CJ Himmelberg, Thiruvenkatachari Parthasarathy, TES Raghavan, and FS Van Vleck. Existence of p-equilibrium and optimal stationary strategies in stochastic games. *Proceedings of the American Mathematical Society*, 60(1):245–251, 1976.

33 Lisa Hutschenreiter, Christel Baier, and Joachim Klein. Parametric Markov Chains: PCTL Complexity and Fraction-free Gaussian Elimination. In *GandALF*, volume 256 of *EPTCS*, pages 16–30, 2017.

34 Nils Jansen, Florian Corzilius, Matthias Volk, Ralf Wimmer, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. Accelerating Parametric Probabilistic Verification. In *QEST*, volume 8657 of *LNCS*, pages 404–420. Springer, 2014.

35 Bengt Jonsson and Kim Guldstrand Larsen. Specification and Refinement of Probabilistic Processes. In *LICS*, pages 266–277. IEEE Computer Society, 1991.

36 Sebastian Junges, Erika Abraham, Christian Hensel, Nils Jansen, Joost-Pieter Katoen, Tim Quatmann, and Matthias Volk. Parameter Synthesis for Markov Models. *CoRR*, abs/1903.07993, 2019. `arXiv:1903.07993`.

37 Sebastian Junges, Nils Jansen, Ralf Wimmer, Tim Quatmann, Leonore Winterer, Joost-Pieter Katoen, and Bernd Becker. Finite-State Controllers of POMDPs using Parameter Synthesis. In *UAI*, pages 519–529. AUAI Press, 2018.

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

39 Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. Parametric probabilistic transition systems for system design and analysis. *Formal Asp. Comput.*, 19(1):93–109, 2007.

40 Arnab Nilim and Laurent El Ghaoui. Robust Control of Markov Decision Processes with Uncertain Transition Matrices. *Operations Research*, 53(5):780–798, 2005.

41 Thiruvenkatachari Parthasarathy. Discounted and positive stochastic games. *Bulletin of the American Mathematical Society*, 77(1):134–136, 1971.

42 Alberto Puggelli, Wenchao Li, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Polynomial-Time Verification of PCTL Properties of MDPs with Convex Uncertainties. In *CAV*, volume 8044 of *LNCS*, pages 527–542. Springer, 2013.

43 Martin L. Puterman. *Markov Decision Processes.* Wiley, 1995.

44 Tim Quatmann, Christian Dehnert, Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. Parameter Synthesis for Markov Models: Faster Than Ever. In *ATVA*, volume 9938 of *LNCS*, pages 50–67, 2016.

45 James Renegar. On the Computational Complexity and Geometry of the First-Order Theory of the Reals, Part I: Introduction. Preliminaries. The Geometry of Semi-Algebraic Sets. The Decision Problem for the Existential Theory of the Reals. *J. Symb. Comput.*, 13(3):255–300, 1992.

46 Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach.* Pearson Education, 2010.

47 Marcus Schaefer. *Realizability of Graphs and Linkages*, pages 461–482. Springer New York, 2013.

48 Marcus Schaefer and Daniel Stefankovic. Fixed Points, Nash Equilibria, and the Existential Theory of the Reals. *Theory Comput. Syst.*, 60(2):172–193, 2017.

49 Roberto Segala and Andrea Turrini. Comparative Analysis of Bisimulation Relations on Alternating and Non-Alternating Probabilistic Models. In *QEST*, pages 44–53. IEEE Computer Society, 2005.

50 Koushik Sen, Mahesh Viswanathan, and Gul Agha. Model-Checking Markov Chains in the Presence of Uncertainties. In *TACAS*, volume 3920 of *LNCS*, pages 394–410. Springer, 2006.

51 Sven Seuken and Shlomo Zilberstein. Formal models and algorithms for decentralized decision making under uncertainty. *Autonomous Agents and Multi-Agent Systems*, 17(2):190–250, 2008.

52 Lloyd S. Shapley. Stochastic Games. *PNAS*, 39(10):1095–1100, 1953.

53 Eilon Solan. Continuity of the value of competitive Markov decision processes. *Journal of Theoretical Probability*, 16(4):831–845, 2003.

54 Jeremy Sproston. Qualitative Reachability for Open Interval Markov Chains. In *RP*, volume 11123 of *LNCS*, pages 146–160. Springer, 2018.

55 Richard S. Sutton and Andrew G. Barto. *Reinforcement learning - an introduction.* Adaptive computation and machine learning. MIT Press, 1998.

56 Nikos Vlassis, Michael L. Littman, and David Barber. On the Computational Complexity of Stochastic Controller Optimization in POMDPs. *TOCT*, 4(4):12:1–12:8, 2012.

57 Wolfram Wiesemann, Daniel Kuhn, and Berç Rustem. Robust Markov Decision Processes. *Math. Oper. Res.*, 38(1):153–183, 2013.

58 Di Wu and Xenofon D. Koutsoukos. Reachability analysis of uncertain systems using bounded-parameter Markov decision processes. *Artif. Intell.*, 172(8-9):945–954, 2008.

Timed Basic Parallel Processes

Lorenzo Clemente
University of Warsaw, Poland

Piotr Hofman
University of Warsaw, Poland

Patrick Totzke
University of Liverpool, UK

Abstract

Timed basic parallel processes (TBPP) extend communication-free Petri nets (aka. BPP or commutative context-free grammars) by a global notion of time. TBPP can be seen as an extension of timed automata (TA) with context-free branching rules, and as such may be used to model networks of independent timed automata with process creation. We show that the coverability and reachability problems (with unary encoded target multiplicities) are PSPACE-complete and EXPTIME-complete, respectively. For the special case of 1-clock TBPP, both are NP-complete and hence not more complex than for untimed BPP. This contrasts with known super-Ackermannian-completeness and undecidability results for general timed Petri nets. As a result of independent interest, and basis for our NP upper bounds, we show that the reachability relation of 1-clock TA can be expressed by a formula of polynomial size in the existential fragment of linear arithmetic, which improves on recent results from the literature.

2012 ACM Subject Classification Theory of computation → Timed and hybrid models

Keywords and phrases Timed Automata, Petri Nets

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.15

Related Version An accompanying technical report is available at https://arxiv.org/abs/1907.01240 [17].

Funding *Lorenzo Clemente*: Partially supported by Polish NCN grant 2017/26/D/ST6/00201.
Piotr Hofman: Partially supported by Polish NCN grant 2016/21/D/ST6/01368.

Acknowledgements Many thanks to Rasmus Ibsen-Jensen for helpful discussions and pointing us towards [29].

1 Introduction

We study safety properties of unbounded networks of timed processes, where time is global and elapses at the same rate for every process. Each process is a *timed automaton* (TA) [7] controlling its own set of private clocks, not accessible to the other processes. A process can dynamically create new sub-processes, which are thereafter independent from each other and their parent, and can also terminate its execution and disappear from the network.

While such systems can be conveniently modelled in *timed Petri nets* (TdPN), verification problems for this model are either undecidable or prohibitively complex: The reachability problem is undecidable even when individual processes carry only one clock [41] and the coverability problem is undecidable for two or more clocks. In the one-clock case coverability remains decidable but its complexity is hyper-Ackermannian [6, 28].

These hardness results however require unrestricted synchronization between processes, which motivates us to study of the communication-free fragment of TdPN, called *timed basic parallel processes* (TBPP) in this paper. This model subsumes both TA and communication-

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 15; pp. 15:1–15:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

free Petri nets (a.k.a. BPP [14, 24]). The general picture that we obtain is that extending communication-free Petri nets by a global notion of time comes at no extra cost in the complexity of safety checking, and it improves on the prohibitive complexities of TdPN.

Our contributions. We show that the TBPP coverability problem is PSPACE-complete, matching same complexity for TA [7, 25], and that the more general TBPP reachability problem is EXPTIME-complete, thus improving on the undecidability of TdPN. The lower bounds already hold for TBPP with two clocks if constants are encoded in binary; EXPTIME-hardness for reachability with no restriction on the number of clocks holds for constants in $\{0, 1\}$. The upper bounds are obtained by reduction to TA reachability and reachability games [30], and assume that process multiplicities in target configurations are given in unary.

In the single-clock case, we show that both TBPP coverability and reachability are NP-complete, matching the same complexity for (untimed) BPP [24]. This paves the way for the automatic verification of unbounded networks of 1-clock timed processes, which is currently lacking in mainstream verification tools such as UPPAAL [34] and KRONOS [46]. The NP lower bound already holds when the target configuration has size 2; when it has size one, 1-clock TBPP coverability becomes NL-complete, again matching the same complexity for 1-clock TA [33] (and we conjecture that 1-clock reachability is in PTIME under the same restriction).

As a contribution of independent interest, we show that the ternary reachability relation of 1-clock TA can be expressed by a formula of existential linear arithmetic (∃LA) of polynomial size. By *ternary reachability relation* we mean the family of relations $\{\rightarrow_{pq}\}$ s.t. $\mu \xrightarrow{\delta}_{pq} \nu$ holds if from control location p and clock valuation $\mu \in \mathbb{R}^k_{\geq 0}$ it is possible to reach control location q and clock valuation $\nu \in \mathbb{R}^k_{\geq 0}$ in exactly $\delta \in \mathbb{R}_{\geq 0}$ time. This should be contrasted with analogous results (cf. [27]) which construct formulas of exponential size, even in the case of 1-clock TA. Since the satisfiability problem for ∃LA is decidable in NP, we obtain a NP upper bound to decide ternary reachability $\rightarrow_{pq}$. We show that the logical approach is optimal by providing a matching NP lower bound for the same problem. Our NP upper bounds for the 1-clock TBPP coverability and reachability problems are obtained as an application of our logical expressibility result above, and the fact that ∃LA is in NP; as a further technical ingredient we use polynomial bounds on the piecewise-linear description of value functions in 1-clock priced timed games [29].

Related research. Starting from the seminal PSPACE-completeness result of the nonemptiness problem for TA [7] (cf. also [25]), a rich literature has emerged considered more challenging verification problems, including the symbolic description of the reachability relation [20, 22, 31, 23, 27]. There are many natural generalizations of TA to add extra modelling capabilities, including *time Petri Nets* [36, 38] (which associate timing constraints to transitions) the already mentioned timed Petri nets (TdPN) [41, 6, 28] (where tokens carry clocks which are tested by transitions), *networks of timed processes* [5], several variants of *timed pushdown automata* [12, 21, 8, 43, 2, 40, 9, 19, 18], *timed communicating automata* [32, 16, 4, 15], and their lossy variant [1], and timed process calculi based on Milners CCS (e.g. [10]). While decision problems for TdPN have prohibitive complexity/are undecidable, it has recently been shown that structural safety properties are PSPACE-complete using forward accelerations [3].

Outline. In Section 2 we define TBPP and their reachability and coverability decision problems. In Section 3 we show that the reachability relation for 1-clock timed automata can be expressed in polynomial time in an existential formula of linear arithmetic, and that the latter logic is in NP. We apply this result in Sec. 4 to show that the reachability and coverability problems for 1-clock TBPP is NP-complete. Finally, in Section 5 we study the case of TBPP with $k \geq 2$ clocks, and in Section 6 we draw conclusions. Full proofs can be found in the technical report [17].

2 Preliminaries

Notations. We use $\mathbb{N}$ and $\mathbb{R}_{\geq 0}$ to denote the sets of nonnegative integers and reals, respectively. For $c \in \mathbb{R}_{\geq 0}$ we write $\mathit{int}(c) \in \mathbb{N}$ for its integer part and $\mathit{frac}(c) \overset{\mathit{def}}{=} c - \mathit{int}(c)$ for its fractional part. For a set $\mathcal{X}$, we use $\mathcal{X}^*$ to denote the set of finite sequences over $\mathcal{X}$ and $\mathcal{X}^\oplus$ to denote the set of finite multisets over $\mathcal{X}$, i.e., functions $\mathbb{N}^\mathcal{X}$. We denote the empty multiset by $\emptyset$, we denote the union of two multisets $\alpha, \beta \in \mathbb{N}^\mathcal{X}$ by $\alpha + \beta$, which is defined point-wise, and by $\alpha \leq \beta$ we denote the natural partial order on multisets, also defined point-wise. The *size* of a multiset $\alpha \in \mathbb{N}^\mathcal{X}$ is $|\alpha| = \sum_{X \in \mathcal{X}} \alpha(X)$. We overload notation and we let $X \in \mathcal{X}$ denote the singleton multiset of size 1 containing element X. For example, if $X, Y \in \mathcal{X}$, then the multiset consisting of 1 occurrence of X and 2 of Y will be denoted by $\alpha = X + Y + Y$; it has size $|\alpha| = 3$.

Clocks. Let $\mathcal{C}$ be a finite set of *clocks*. A *clock valuation* is a function $\mu \in \mathbb{R}_{\geq 0}^\mathcal{C}$ assigning a nonnegative real to every clock. For $t \in \mathbb{R}_{\geq 0}$, we write $\mu + t$ for the valuation that maps clock $x \in \mathcal{C}$ to $\mu(x) + t$. For a clock $x \in \mathcal{C}$ and a clock or constant $e \in \mathcal{C} \cup \mathbb{N}$ let $\mu[x := e]$ be the valuation ν s.t. $\nu(x) = \mu(e)$ and $\nu(z) = \mu(z)$ for every other clock $z \neq x$ (where we assume $\mu(k) = k$ for a constant $k \in \mathbb{N}$); for a sequence of assignments $R = (x_1 := e_1; \cdots; x_n := e_n)$ let $\mu[R] = \mu[x_1 := e_1] \cdots [x_n := e_n]$. A *clock constraint* is a conjunction of linear inequalities of the form $c \bowtie k$, where $c \in \mathcal{C}$, $k \in \mathbb{N}$, and $\bowtie \in \{<, \leq, =, \geq, >\}$; we also allow **true** for the trivial constraint which is always satisfied. We write $\mu \models \varphi$ to denote that the valuation μ satisfies the constraint φ.

Timed basic parallel processes. A *timed basic parallel process* (TBPP) consists of finite sets $\mathcal{C}$, $\mathcal{X}$, and $\mathcal{R}$ of clocks, nonterminal symbols, and rules. Each rule is of the form

$$X \overset{\varphi; R}{\Longrightarrow} \alpha$$

where $X \in \mathcal{X}$ is a nonterminal, φ is a clock constraint, R is a sequence of assignments of the form $x := e$, where e is either a constant in $\mathbb{N}$ or a clock in $\mathcal{C}$, and $\alpha \in \mathcal{X}^\oplus$ is a finite multiset of successor nonterminals[1]. Whenever the test $\varphi \equiv$ **true** is trivial, or R is the empty sequence, we just omit the corresponding component and just write $X \overset{\varphi}{\Longrightarrow} \alpha$, $X \overset{R}{\Longrightarrow} \alpha$, or $X \Longrightarrow \alpha$. Finally, we say that we *reset* the clock x_i if we assign it to 0.

Henceforth, we assume w.l.o.g. that the size $|\alpha|$ is at most 2. A rule with $\alpha = \emptyset$ is called a *vanishing* rule, and a rule with $|\alpha| = 2$ is called a *branching* rule. We will write k-TBPP to denote the class of TBPP with k clocks.

A *process* is a pair $(X, \mu) \in \mathcal{X} \times \mathbb{R}_{\geq 0}^\mathcal{C}$ comprised of a nonterminal X and a clock valuation μ, and a *configuration* α is a multiset of processes, i.e., $\alpha \in (\mathcal{X} \times \mathbb{R}_{\geq 0}^\mathcal{C})^\oplus$. For a process $P = (X, \mu)$ and $t \in \mathbb{R}_{\geq 0}$, we denote by $P + t$ the process $(X, \mu + t)$, and for a

[1] We note that clock updates $x := k$ with $k \in \mathbb{N}$ can be encoded with only a polynomial blow-up by replacing them with $x := 0$, while recording in the finite control the last update k, and replacing a test $x \bowtie h$ with $x \bowtie h - k$. We use them as a syntactic sugar to simplify the presentation of some constructions.

configuration $\alpha = P_1 + \cdots + P_n$, we denote by $\alpha + t$ the configuration $Q_1 + \cdots + Q_n$, where $Q_1 = P_1 + t, \ldots, Q_n = P_n + t$. The semantics of a TBPP $(\mathcal{C}, \mathcal{X}, \mathcal{R})$ is given by an infinite timed transition system $(C, \rightarrow)$, where $C = (\mathcal{X} \times \mathbb{R}^{\mathcal{C}}_{\geq 0})^{\oplus}$ is the set of configurations, and $\rightarrow \subseteq C \times \mathbb{R}_{\geq 0} \times C$ is the transition relation between configurations. There are two kinds of transitions:

Time elapse: For every configuration $\alpha \in C$ and $t \in \mathbb{R}_{\geq 0}$, there is a transition $\alpha \xrightarrow{t} \alpha + t$ in which all clocks in all processes are simultaneously increased by t. In particular, the empty configuration stutters: $\emptyset \xrightarrow{t} \emptyset$, for every $t \in \mathbb{R}_{\geq 0}$.

Discrete transitions: For every configuration $\gamma = \alpha + (X, \mu) + \beta \in C$ and rule $X \overset{\varphi; R}{\Longrightarrow} Y + Z$ s.t. $\mu \models \varphi$ there is a transition $\gamma \xrightarrow{0} \alpha + (Y, \nu) + (Z, \nu) + \beta$, where $\nu = \mu[R]$. Analogously, rules $X \overset{\varphi; R}{\Longrightarrow} Y$ and $X \overset{\varphi; R}{\Longrightarrow} \emptyset$ induce transitions $\gamma \xrightarrow{0} \alpha + (Y, \nu) + \beta$ and $\gamma \xrightarrow{0} \alpha + \beta$.

A *run* starting in α and ending in β is a sequence of transitions $\alpha = \alpha_0 \xrightarrow{t_1} \alpha_1 \cdots \xrightarrow{t_n} \alpha_n = \beta$. We write $\alpha \xrightarrow{t} \beta$ whenever there is a run as above where the sum of delays is $t = t_1 + \cdots + t_n$, and we write $\alpha \xrightarrow{*} \beta$ whenever $\alpha \xrightarrow{t} \beta$ for some $t \in \mathbb{R}_{\geq 0}$.

TBPP generalise several known models: A *timed automaton* (TA) [7] is a TBPP without branching rules; in the context of TA, we will sometimes call nonterminals with the more standard name of *control locations*. Untimed *basic parallel processes* (BPP) [14, 24] are TBPP over the empty set of clocks $\mathcal{X} = \emptyset$. TBPP can also be seen as a structural restriction of *timed Petri nets* [6, 28] where each transition consumes only one token at a time.

TBPP are related to *alternating timed automata* (ATA) [37, 35]: Branching in TBPP rules corresponds to universal transitions in ATA. However, ATA offer additional means of synchronisation between the different branches of a run tree: While in a TBPP synchronisation is possible only through the elapse of time, in an ATA all branches must read the same timed input word.

Decision problems. We are interested in checking safety properties of TBPP in the form of the following decision problems. The *reachability problem* asks whether a target configuration is reachable from a source configuration.

> **Input**: A TBPP $(\mathcal{C}, \mathcal{X}, \mathcal{R})$, an initial $X \in \mathcal{X}$ and target nonterminals $T_1, \ldots, T_n \in \mathcal{X}$.
> **Question**: Does $(X, \vec{0}) \xrightarrow{*} (T_1, \vec{0}) + \cdots + (T_n, \vec{0})$ hold?

It is crucial that we reach all processes in the target configurations *at the same time*, which provides an external form of global synchronisation between processes.

Motivated both by complexity considerations and applications for safety checking, we study the *coverability problem*, where it suffices to reach some configuration larger than the given target in the multiset order. For configurations $\alpha, \beta \in (\mathcal{X} \times \mathbb{R}^{\mathcal{C}}_{\geq 0})^{\oplus}$, let $\alpha \xrightarrow{*} \cdot \geq \beta$ whenever there exists $\gamma \in (\mathcal{X} \times \mathbb{R}^{\mathcal{C}}_{\geq 0})^{\oplus}$ s.t. $\alpha \xrightarrow{*} \gamma \geq \beta$.

> **Input**: A TBPP $(\mathcal{C}, \mathcal{X}, \mathcal{R})$, an initial $X \in \mathcal{X}$ and target nonterminals $T_1, \ldots, T_n \in \mathcal{X}$.
> **Question**: Does $(X, \vec{0}) \xrightarrow{*} \cdot \geq (T_1, \vec{0}) + \cdots + (T_n, \vec{0})$?

The *simple* reachability/coverability problems are as above but with the restriction that the target configuration is of size 1, i.e., a single process. Notice that this is a proper restriction, since reachability and coverability do not reduce in general to their simple variant. Finally, the *non-emptiness problem* is the special case of the reachability problem where the target configuration α is the empty multiset $\emptyset$.

In all decision problems above the restriction to zero-valued clocks in the initial process is mere convenience, since we could introduce a new initial nonterminal Y and a transition $Y \xRightarrow{x_1:=\mu(x_1);\ldots;x_n:=\mu(x_n)} X$ initialising the clocks to the initial values provided by the (rational) clock valuation $\mu \in \mathbb{Q}_{\geq 0}^{\mathcal{C}}$. Similarly, if we wanted to reach the final configuration $\alpha = (X_1, \mu_1)+(X_2, \mu_2)$ with $\mu_1, \mu_2 \in \mathbb{Q}_{\geq 0}^{\mathcal{C}}$, then we could add two nonterminals Y_1, Y_2 and two new rules $X_1 \xRightarrow{x_1=\mu_1(x_1)\wedge\cdots\wedge x_n=\mu_1(x_n);x_1:=0;\ldots;x_n:=0} Y_1$ and $X_2 \xRightarrow{x_1=\mu_2(x_1)\wedge\cdots\wedge x_n=\mu_2(x_n);x_1:=0;\ldots;x_n:=0} Y_2$ and check whether $X \xrightarrow{*} (Y_1, \vec{0}) + (Y_2, \vec{0})$ holds. (It is standard to transform TBPP with constraints of the form $x_i = k$ with $k \in \mathbb{Q}_{\geq 0}$ in the form $x_i = k$ with $k \in \mathbb{N}$.) Similarly, the restriction of having just one initial nonterminal process is also w.l.o.g., since if we wanted to check reachability from $(X_1, \vec{0}) + (X_2, \vec{0})$ we could just add a new initial nonterminal X and a branching rule $X \Longrightarrow X_1 + X_2$.

For complexity considerations we will assume that all constants appearing in clock constraints are given in binary encoding, and that the multiplicities of target processes are in unary.

3 Reachability Relations of One-Clock Timed Automata

In this section we show that the reachability relation of 1-clock TA is expressible as an existential formula of linear arithmetic of polynomial size. Since the latter fragment is in NP, this gives an NP algorithm to check whether a family of TA can reach the respective final locations *at the same time*. This result will be applied in Sec. 4 to show that coverability and reachability of 1-TBPP are in NP. We first show that existential linear arithmetic is in NP (which is an observation of independent interest), and then how to express the reachability relation of 1-TA in existential linear arithmetic in polynomial time.

The set of *terms* t is generated by the following abstract grammar

$$s, t \ ::= \ x \mid k \mid \lfloor t \rfloor \mid \mathit{frac}(t) \mid -t \mid s + t \mid k \cdot t,$$

where x is a rational variable, $k \in \mathbb{Z}$ is an integer constant encoded in binary, $\lfloor t \rfloor$ represents the integral part of t, and $\mathit{frac}(t)$ its fractional part. *Linear arithmetic* (LA) is the first order language with atomic proposition of the form $s \leq t$ [45], we denote by ∃LA its existential fragment, and by qf-LA its quantifier-free fragment. Linear arithmetic generalises both *Presburger arithmetic* (PA) and *rational arithmetic* (RA), whose existential fragments are known to be in NP [39, 26]. This can be generalised to ∃LA. (The same result can be derived from the analysis of [11, Theorem 3.1]).

▶ **Theorem 1.** *The existential fragment ∃LA of LA is in* NP.

Let $\mathcal{A} = (\mathcal{C}, \mathcal{X}, \mathcal{R})$ be a k-TA. The *ternary reachability relation* of $\mathcal{A}$ is the family of relations $\{\rightarrow_{XY}\}_{X,Y\in\mathcal{X}}$, where each $\rightarrow_{XY} \subseteq \mathbb{R}_{\geq 0}^{\mathcal{C}} \times \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0}^{\mathcal{C}}$ is defined as: $\mu \xrightarrow{\delta}_{XY} \nu$ iff $(X, \mu) \xrightarrow{\delta} (Y, \nu)$. We say that the reachability relation is *expressed* by a family of LA formulas $\{\varphi_{XY}\}_{X,Y\in\mathcal{X}}$ if

$$\mu \xrightarrow{\delta}_{XY} \nu \quad \text{iff} \quad (\mu, \delta, \nu) \models \varphi_{XY}(\vec{x}, t, \vec{y}), \text{ for every } X, Y \in \mathcal{X}, \mu, \nu \in \mathbb{R}_{\geq 0}^{\mathcal{C}}, \delta \in \mathbb{R}_{\geq 0}.$$

In the formula $\varphi_{XY}(\vec{x}, t, \vec{y})$, $\vec{x}$ are k variables representing the clock values in location X at the beginning of the run, $\vec{y}$ are k variables representing the clock values in location Y at the end of the run, and t is a single variable representing the total time elapsed during the run. In the rest of this section, we assume that the TA has only one clock $\mathcal{X} = \{x\}$.

The main result of this section is that 1-TA reachability relations are expressible by ∃LA formulas constructible in polynomial time.

▶ **Theorem 2.** *Let $\mathcal{A}$ be a 1-TA. The reachability relation $\{\to_{XY}\}_{X,Y\in\mathcal{X}}$ is expressible as a family of formulas $\{\varphi_{XY}\}_{X,Y\in\mathcal{X}}$ of existential linear arithmetic ∃LA in polynomial time.*

In the rest of the section we prove the theorem above. We begin with some preliminaries.

Interval abstraction. We replace the integer value of the clock x by its interval [33]. Let $0 = k_0 < k_1 < \cdots < k_n < k_{n+1} = \infty$ be all integer constants appearing in constraints of $\mathcal{A}$, and let the set of intervals be the following totally ordered set:

$$\Lambda = \{\{k_0\} < (k_0, k_1) < \{k_1\} < \cdots < (k_{n-1}, k_n) < \{k_n\} < (k_n, k_{n+1})\}\,.$$

Clearly, we can resolve any constraint of $\mathcal{A}$ by looking at the interval $\lambda \in \Lambda$. We write $\lambda \models \varphi$ whenever $v \models \varphi$ for some $v \in \lambda$ (whose choice does not matter by the definition of λ).

The construction. Let $\mathcal{A} = (\{x\}, \mathcal{X}, \mathcal{R})$ be a TA. In order to simplify the presentation below, we assume w.l.o.g. that the only clock updates are resets $x := 0$ (cf. footnote 1). We build an NFA $\mathcal{B} = (\Sigma, Q, \to)$ where Σ contains symbols (r, ε) and $(r, \checkmark_\lambda)$ for every transition $r \in \mathcal{R}$ of $\mathcal{A}$ and interval $\lambda \in \Lambda$, and an additional symbol τ representing time elapse, and $Q = \mathcal{X} \times \Lambda$ is a set of states of the form (X, λ), where $X \in \mathcal{X}$ is a control location of $\mathcal{A}$ and $\lambda \in \Lambda$ is an interval. Transitions $\to \subseteq Q \times \Sigma \times Q$ are defined as follows. A rule $r = X \xRightarrow{\varphi;R} Y \in \mathcal{R}$ of $\mathcal{A}$ generates one or more transitions in $\mathcal{B}$ of the form

$$(X, \lambda) \xrightarrow{(r,a)} (Y, \mu)$$

whenever $\lambda \models \varphi$ and any of the following two conditions is satisfied:

- the clock is not reset i.e R is equal $x := x$, and $\mu = \lambda, a = \varepsilon$, or
- the clock is reset $x := 0$, $\mu = \{0\}$, and the automaton emits a tick $a = \checkmark_\lambda$.

A time elapse transition is simulated in $\mathcal{B}$ by transitions of the form

$$(X, \lambda) \xrightarrow{\tau} (X, \mu), \qquad \lambda \leq \mu \text{ (the total ordering on intervals).}$$

Reachability relation of $\mathcal{A}$. For a set of finite words $L \subseteq \Sigma^*$, let $\psi_L(\vec{y})$ be a formula of existential Presburger arithmetic with a free integral variable y_λ for every interval $\lambda \in \Lambda$ counting the number of symbols of the form $(r, \checkmark_\lambda)$, for some $r \in \mathcal{R}$. The formula ψ_L can be computed from the Parikh image of L: By [44, Theorem 4], a formula $\tilde{\psi}_L(\vec{z})$ of existential Presburger arithmetic can be computed in linear time from an NFA (or even a context-free grammar) recognising L, and then one just defines $\psi_L(\vec{y}) \equiv \exists \vec{z} \cdot \tilde{\psi}_L(\vec{z}) \wedge \bigwedge_{\lambda\in\Lambda} y_\lambda = \sum_{r\in\mathcal{R}} z_{r,\lambda}$. Let L_{cd} be the regular language recognised by $\mathcal{B}$ by making c initial and d final, and let $\Lambda = \{\lambda_0, \ldots, \lambda_{2n+1}\}$ contain $2n + 2$ intervals. Let $\psi_{cd}(x, t, x')$ be a formula of existential Presburger arithmetic computing the total elapsed time t, given the initial x and final x' values of the unique clock:

$$\begin{aligned} \psi_{cd}(x, t, x') \equiv{} & \exists y_0, \ldots, y_{2n+1} \cdot \psi_{L_{cd}}(\lfloor y_0 \rfloor, \ldots, \lfloor y_{2n+1} \rfloor) \wedge \\ & \exists z_0, \ldots, z_{2n+1} \cdot \bigwedge_{\lambda_i\in\Lambda} (z_i \in y_i \cdot \lambda_i) \wedge t = x' - x + \sum_{\lambda_i\in\Lambda} z_i, \text{ where} \end{aligned}$$

$$z \in y \cdot \lambda \equiv \begin{cases} a \cdot y < z < b \cdot y & \text{if } \lambda = (a, b), \\ z = a \cdot y & \text{if } \lambda = \{a\}\,. \end{cases}$$

Intuitively, y_i represents the total number of times the clock is reset while in interval λ_i, and z_i represents the sum of the values of the clock when it is reset in interval λ_i. For control locations X, Y of $\mathcal{A}$, let

$$\varphi_{XY}(x,t,x') \equiv \bigvee_{\lambda,\mu\in\Lambda} \{x \in \lambda \wedge x' \in \mu \wedge \psi_{cd}(x,t,x') \mid c = (X,\lambda), d = (Y,\mu)\}.$$

The correctness of the construction is stated below.

▶ **Lemma 3.** *For every configurations (X,u) and (Y,v) of $\mathcal{A}$ and total time elapse $\delta \geq 0$,*

$$u \xrightarrow{\delta}_{XY} v \quad \textit{iff} \quad (u,\delta,v) \models \varphi_{XY}(x,t,x').$$

We conclude this section by applying Theorem 2 to solve the 1-TA ternary reachability problem. The *ternary reachability problem* takes as input a TA $\mathcal{A}$ as above, with two distinguished control locations $X, Y \in \mathcal{X}$, and a total duration $\delta \in \mathbb{Q}$ (encoded in binary), and asks whether $(\vec{0}) \xrightarrow{\delta}_{XY} (\vec{0})$. The result below shows that computing 1-TA reachability relations is optimal in order to solve the ternary reachability problem.

▶ **Theorem 4.** *The ternary reachability problem for 1-TA is* NP*-complete.*

Proof. For the upper bound, apply Theorem 2 to construct in polynomial time a formula of ∃LA expressing the reachability relation and check satisfiability in NP thanks to Theorem 1.

The lower bound can be seen by reduction from SubsetSum. Let $\mathcal{S} = \{a_1,\ldots,a_k\} \subseteq \mathbb{N}$ and $a \in \mathbb{N}$ be the input to the subset sum problem, whereby we look for a subset $\mathcal{S}' \subseteq \mathcal{S}$ s.t. $a = \sum_{b\in\mathcal{S}'} b$. We construct a TA with a single clock x and locations $\mathcal{X} = \{X_0,\ldots,X_k\}$, where X_0 is the initial location and X_k the target. A path through the system describes a subset by spending exactly 0 or a_i time in location X_i (see Figure 1). In the constructed automaton, $(X_1,0) \xrightarrow{a} (X_k,0)$ iff the subset sum instance was positive. ◀

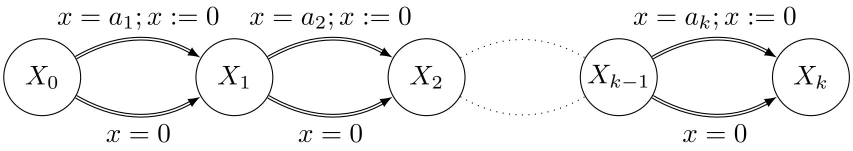


Figure 1 Reduction from subset sum to 1-TA (ternary) reachability. We have $(X_0,0) \xrightarrow{t} (X_k,0)$ iff $t = \sum_{b\in\mathcal{S}'} b$ for some subset $\mathcal{S}' \subseteq \{a_1,\ldots,a_k\}$.

4 One-Clock TBPP

As a warm-up we note that the simple coverability problem for 1-TBPP, where the target has size one, is inter-reducible with the reachability problem for 1-clock TA and hence NL-complete [33].

▶ **Theorem 5.** *The simple coverability problem for 1-clock TBPP is* NL*-complete.*

Proof. The lower bound is trivial since 1-TBPP generalize 1-TA. For the other direction we can transform a given TBPP into a TA by replacing branching rules of the form $X \overset{\varphi,R}{\Longrightarrow} Y + Z$ with two rules $X \overset{\varphi,R}{\Longrightarrow} Y$ and $X \overset{\varphi,R}{\Longrightarrow} Z$. In the constructed TA we have $(X,\mu) \xrightarrow{*} (Y,\nu)$ if, and only if, $(X,\mu) \xrightarrow{*} (Y,\nu) + \gamma$ for some γ in the original TBPP. ◀

The construction above works because the target is a single process and so there are no constraints on the other processes in γ, which were produced as side-effects by the branching rules. The (non-simple) 1-TBPP coverability problem is in fact NP-complete. Indeed, even for untimed BPP, coverability is NP-hard, and this holds already when target sets are encoded in unary (which is the setting we are considering here) [24]. We show that for 1-TBPP this lower bound already holds if the target has fixed size 2.

▶ **Lemma 6.** *Coverability is* NP*-hard for 1-TBPP already for target sets of size* ≥ 2*.*

Proof. We proceed by reduction from subset sum as in Theorem 4. The only difference will be that here, we use one extra process to keep track of the total elapsed time. Let $\mathcal{S} = \{a_1, \ldots, a_k\} \subseteq \mathbb{N}$ and $t \in \mathbb{N}$ be the input to the subset sum problem. We construct a 1-TBPP with nonterminals $\mathcal{X} = \{S, X_0, \ldots, X_k, Y\}$, where S is the initial nonterminal and Y will be used to keep track of the total time elapsed. The rules are as in the proof of Theorem 4, and additionally we have an initial branching rule $S \overset{x=0}{\Longrightarrow} X_0 + Y$. We have $(S, 0) \xrightarrow{*} \cdot \geq (X_k, 0) + (Y, t)$ if, and only if, $t = \sum_{b \in \mathbb{B}} b$ for some subset $\mathcal{S}' \subseteq \mathcal{S}$. ◀

In the remainder of this section we will argue (Theorem 11) that a matching NP upper bound even holds for the *reachability* problem for 1-TBPP. Let us first motivate the key idea behind the construction. Consider the following TBPP coverability query:

$$(S, 0) \xrightarrow{*} \cdot \geq (A, 0) + (B, 0). \tag{†}$$

If (†) holds, then there is a derivation tree witnessing that $(S, 0) \xrightarrow{*} (A, 0) + (B, 0) + \gamma$ for some configuration γ. The least common ancestor of leaves $(A, 0)$ and $(B, 0)$ is some process $(C, c) \in (\mathcal{X} \times \mathbb{R}_{\geq 0})$. Consider the TA $\mathcal{A}$ obtained from the TBPP by replacing branching rules $X \overset{\varphi;R}{\Longrightarrow} X_i + X_j$ with linear rules $X \overset{\varphi;R}{\Longrightarrow} X_i$ and $X \overset{\varphi;R}{\Longrightarrow} X_j$, and let the reachability relation of $\mathcal{A}$ be expressed by $\exists$LA formulas $\{\varphi_{XY}\}_{X,Y \in \mathcal{X}}$, which are of polynomial size by Theorem 2. Then our original coverability query (†) is equivalent to satisfiability the following $\exists$LA formula:

$$\psi \;\equiv\; \exists t_0, t_1, c \in \mathbb{R} \cdot (\varphi_{SC}(0, t_0, c) \;\wedge \varphi_{CA}(c, t_1, 0) \wedge \varphi_{CB}(c, t_1, 0)) \,.$$

More generally, for any coverability query $(S, 0) \xrightarrow{*} \cdot \geq \alpha$ the number of common ancestors is linear in $|\alpha|$, and thus we obtain a $\exists$LA formula ψ of polynomial size, whose satisfiability we can check in NP thanks to Theorem 1.

▶ **Theorem 7.** *The coverability problem for 1-clock TBPP is* NP*-complete.*

In order to witness reachability instances we need to refine the argument above to restrict the TA in such a way that they do not accidentally produce processes that cannot be removed in time. To illustrate this point, consider a 1-TBPP with rules

$$X \overset{x=0}{\Longrightarrow} Y + Z \qquad \text{and} \qquad Z \overset{x>0}{\Longrightarrow} \emptyset.$$

Clearly $(X, 0) \xrightarrow{*} (Y, 0)$ holds in the TA with rules $X \overset{x=0}{\Longrightarrow} Y$ and $X \overset{x=0}{\Longrightarrow} Z$ instead of the branching rule above. In the TBPP however, $(X, 0)$ cannot reach $(Y, 0)$ because the branching rule produces a process $(Z, 0)$, which needs a positive amount of time to be rewritten to $\emptyset$.

▶ **Definition 8.** *For a nonterminal X let $Vanish_X \subseteq \mathbb{R}^2$ be the binary predicate such that*

$$Vanish_X(x,t) \quad \textit{if} \quad (X,x) \xrightarrow{t} \emptyset$$

Intuitively, $Vanish_X(x,t)$ holds if the configuration (X,x) can vanish in time at most t.

The time it takes to remove a processes (Z,z) can be computed as the value of a one-clock priced timed game [13, 29]. These are two-player games played on 1-clock TA where players aim to minimize/maximize the cost of a play leading up to a designated target state. Nonnegative costs may be incurred either by taking transitions, or by letting time elapse. In the latter case, the incurred cost is a linear function of time, determined by the current control-state. Bouyer et al. [13] prove that such games admit ε-optimal strategies for both players, so have well-defined cost value functions determining the best cost as a function of control-states and clock valuation. They prove that these value functions are in fact piecewise-linear. Hansen et al. [29] later show that the piecewise-linear description has only polynomially many line segments and can be computed in polynomial time[2]. We derive the following lemma.

▶ **Lemma 9.** *A qf-LA formula expressing $Vanish_X$ is effectively computable in polynomial time. More precisely, there is a set $\mathcal{I}$ of polynomially many consecutive intervals $\{a_0\}(a_0,a_1)\{a_1\}(a_1,a_2)\{a_2\},\ldots(a_k,\infty)$ so that*

$$Vanish_X(x,t) \;\equiv\; \bigvee_{0\leq i\leq k} (x = a_i \wedge t \geq_i c_i) \;\vee\; (a_i < x < a_{i+1} \wedge t \geq_i c_i - b_i x),$$

where the $a_i, c_i \in \mathbb{R}$ can be represented using polynomially many bits, $\geq_i \in \{\geq, >\}$ and $b_i \in \{0,1\}$, for all $0 \leq i < k$.

Proof (Sketch). One can construct a one-clock priced timed game in which minimizer's strategies correspond to derivation trees. To do this, let unary rules $X \overset{\varphi;R}{\Longrightarrow} Y$ carry over as transitions between (minimizer) states X, Y; vanishing rules $X \overset{\varphi}{\Longrightarrow} \emptyset$ are replaced by transitions leading to a new target state $\bot$, which has a clock-resetting self-loop. Branching rules $X \overset{\varphi;R}{\Longrightarrow} Y + Z$ can be implemented by rules $X \overset{\varphi}{\Longrightarrow} [Y,Z,\varphi]$, $[Y,Z,\varphi] \overset{\varphi;R}{\Longrightarrow} Y$ and $[Y,Z,\varphi] \overset{\varphi;R}{\Longrightarrow} Z$, where X,Y,Z are minimizer states and $[Y,Z,\varphi]$ is a maximizer state. The cost of staying in a state is 1, transitions carry no costs. Moreover, we need to prevent maximizer from elapsing time from the states she controls. For this reason, we consider an extension of price timed games where maximizer cannot elapse time. In the constructed game, minimizer has a strategy to reach $(\bot,0)$ from (X,x) at cost t iff $(X,x) \xrightarrow{t} \emptyset$. The result now follows from [29, Theorem 4.11] (with minor adaptations in order to consider the more restrictive case where maximizer cannot elapse time) that computes value functions for the cost of reachability in priced timed games. These are piecewise-linear with only polynomially many line segments of slopes 0 or 1 which allows to present $Vanish_X$ in qf-LA as stated. ◀

Lemma 9 allows us to compute a polynomial number of intervals $\mathcal{I}$ sufficient to describe the $Vanish_X$ predicates. We will call a pair $(X,I) \in \mathcal{X} \times \mathcal{I}$ a *region* here. A crucial ingredient for our construction will be timed automata that are restricted in which regions they are allowed to produce as side-effects. To simplify notations let us assume w.l.o.g. that the given

[2] This observation was already made, without proof, in [42, Sec. 7.2.2].

TBPP has no resets along branching rules. Let $I(r) \in \mathcal{I}$ denote the unique interval containing $r \in \mathbb{R}_{\geq 0}$, and for a subset $S \subseteq \mathcal{X} \times \mathcal{I}$ of regions write "$Z \in S$" for the clock constraint expressing that $(Z, I(x)) \in S$. More precisely,

$$Z \in S \;\equiv \bigvee_{(Z,(a_i,a_{i+1}))\in S} a_i < x < a_{i+1} \;\vee \bigvee_{(Z,\{a_i\})\in S} a_i = x.$$

▶ **Definition 10.** *Let $S \subseteq \mathcal{X} \times \mathcal{I}$ be a set of regions for the 1-TBPP $(\{x\}, \mathcal{X}, \mathcal{R})$. We define a timed automaton $TA_S = (\{x\}, \mathcal{X}, \mathcal{R}_S)$ so that $\mathcal{R}_S$ contains all of the rules in $\mathcal{R}$ with rhs of size 1 and none of the vanishing rules. Moreover, every branching rule $X \overset{\varphi}{\Longrightarrow} Y + Z$ in $\mathcal{R}$ introduces*

- *a rule $X \overset{\varphi'}{\Longrightarrow} Y$ guarded by $\varphi' \overset{def}{=} \varphi \wedge Z \in S$, and*
- *a rule $X \overset{\varphi'}{\Longrightarrow} Z$ guarded by $\varphi' \overset{def}{=} \varphi \wedge Y \in S$.*

▶ **Theorem 11.** *The reachability problem for 1-clock TBPP is in* NP*.*

Proof. Suppose that there is a derivation tree witnessing a positive instance of the reachability problem and so that all branches leading to targets have duration τ. We can represent a node by a triple $(A, a, \hat{a}) \in (\mathcal{X} \times \mathbb{R} \times \mathbb{R})$, where (A, a) is a TBPP process and the third component $\hat{a}$ is the total time elapsed so far. Call a node $(A, a, \hat{a})$ *productive* if it lies on a branch from root to some target node. Naturally, every node $(A, a, \hat{a})$ has a unique region $(A, I(a))$ associated with it. For a productive node let us write

$$S(A, a, \hat{a})$$

for the set of regions of nodes which are descendants of $(A, a, \hat{a})$ which are non-productive but have a productive parent. See Figure 2 (left) for an illustration. Observe that

1. The sets $S(A, a, \hat{a})$ can only decrease along a branch from root to a target.
2. If $(A, a, \hat{a})$ has a productive descendant $(C, c, \hat{c})$ such that $S(A, a, \hat{a}) = S(C, c, \hat{c})$, then $(A, a) \xrightarrow{\hat{c}-\hat{a}} (C, c)$ in the timed automaton TA_S.
3. Suppose $(A, a, \hat{a})$ has only one productive child $(C, c, \hat{c})$ and that $S(A, a, \hat{a}) \supset S(C, c, \hat{c})$. Then it must also have another child $(B, b, \hat{b})$ s.t. $Vanish_B(b, \tau - \hat{b})$ holds.

The first two conditions are immediate from the definitions of S and TA_S. To see the third, note that the $S(A, a, \hat{a}) \supset S(C, c, \hat{c})$ implies that $(A, a, \hat{a})$ has some non-productive descendant $(B, b, \hat{b})$ whose region $(B, I(b))$ is not in $S(C, c, \hat{c})$. Since $(C, c, \hat{c})$ is the only productive child, that descendant must already be a child of $(A, a, \hat{a})$. Finally, observe that every non-productive node $(B, b, \hat{b})$ satisfies $Vanish_B(b, \tau - \hat{b})$, as otherwise one of its descendants is present at time τ, and thus must be a target node, contradicting the non-productivity assumption.

The conditions above allow us to use labelled trees of polynomial size as reachability witnesses: These witnesses are labelled trees as above where only as polynomial number of checkpoints along branches from root to target are kept: A checkpoint is either the least common ancestor of two target nodes (in which case a corresponding branching rule must exist), or otherwise it is a triple of nodes as described by condition (3), where a region $(B, I(b))$ is produced for the last time. The remaining paths between checkpoints are positive reachability instances of timed automata TA_S, as in condition (2), where the bottom-most automata TA_S satisfy that if $(U, I) \in S$ then $(z, 0) \in Vanish_U$ for all $z \in I$. Cf. Figure 2 (right). Notice that the existence of a witness of this form is expressible as a polynomially large ∃LA formula thanks to Lemma 9 and Theorem 2.

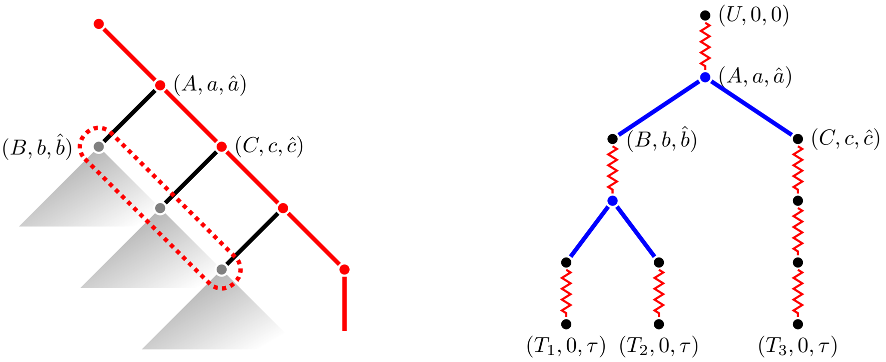


Figure 2 Left: Nodes on the red branch are productive, grey sub-trees are non-productive. $S(A, a, \hat{a})$ contains the regions of nodes in the dotted region. It holds that $S(A, a, \hat{a}) \supseteq S(C, c, \hat{c})$ and the inequality is strict iff $(B, I(b)) \in S(C, c, \hat{c})$. **Right:** small reachability witnesses contain checkpoint where two productive branches split (in blue) or where the allowed side-effects S strictly decrease (red). The intermediate paths are runs of S-restricted TA.

Clearly, every full derivation tree gives rise to a witness of this form. Conversely, assume a witness tree as above exists. One can build a partial derivation tree by unfolding all intermediate TA paths between consecutive checkpoints. It remains to show that whenever some TA_S uses a rule $A \overset{\varphi}{\Longrightarrow} C$ originating from a TBPP rule $A \overset{\varphi}{\Longrightarrow} B + C$ to produce a productive node $(C, c, \hat{c})$ then the node $(B, b, \hat{b})$ produced as side-effect can vanish in time $\tau - \hat{b}$, i.e., we have to show that then $Vanish_B(b, \tau - \hat{b})$.

W.l.o.g. let $Vanish_B(x, t) \equiv t \geq d - xf$ (the case with $>$ is analogous) for some $d \in \mathbb{R}_{\geq 0}$ and $f \in \{0, 1\}$. Observe that the region $(B, I(b))$ is in S by definition of TA_S and that the witness contains a later node $(B, b', \hat{b}')$ with $Vanish_B(b', \tau - \hat{b}')$, and thus

$$\tau - \hat{b}' \geq d - b'f.$$

Notice also that $b' \geq b + \hat{b}' - \hat{b}$ as in the worst-case no reset appears on the path between the parent of $(B, b, \hat{b})$ and $(B, b', \hat{b}')$. Together with the inequality above we derive that $\tau - \hat{b} \geq d - bf$, meaning that indeed $Vanish_B(b, \tau - \hat{b})$ holds, as required. ◀

5 Multi-Clock TBPP

In this section we consider the complexities of coverability and reachability problems for TBPP with multiple clocks. For the upper bounds we will reduce to the reachability problem for TA [7] and to solving reachability games for TA [30].

▶ **Theorem 12.** *The coverability problem for k-TBPP with $k \geq 2$ clocks is* PSPACE*-complete.*

Proof. The lower bound already holds for the reachability problem of 2-clock TA [25] and hence for the *simple* TBPP coverability. For the upper bound, consider an instance where $\mathcal{A} = (\mathcal{C}, \mathcal{X}, \mathcal{R})$ is a k-TBPP and $T_1, \ldots, T_m$ are the target nonterminals. We reduce to the reachability problem for TA $\mathcal{B} = (\mathcal{C}', \mathcal{X}', \mathcal{R}')$ with exponentially many control states $\mathcal{X}'$, but only $|\mathcal{C}'| = O(k \cdot |\mathcal{X}| \cdot m)$ many clocks. The result then follows by the classical region construction of [7], which requires space logarithmic in the number of nonterminals and polynomial in the number of clocks. The main idea of this construction is to introduce (exponentially many) new nonterminals and rules to simulate the original behaviour on bounded configurations only.

Let $n = m + 2$. We have a clock $x_{X,i} \in \mathcal{C}'$ for every original clock $x \in \mathcal{C}$, nonterminal $X \in \mathcal{X}$, and index $1 \leq i \leq n$, and a nonterminal of the form $[\alpha] \in \mathcal{R}'$ for every multiset $\alpha \in \mathcal{X}^{\oplus}$ of size at most $|\alpha| \leq n$. Since we are solving the coverability problem, we do not need to address vanishing rules $X \overset{\varphi;R}{\Longrightarrow} \emptyset$ in $\mathcal{R}$, which are ignored. We will use clock assignments $S_{X,i} \equiv \bigwedge_{i \leq j \leq n-1} x_{X,j} := x_{X,j+1}$ shifting by one position the clocks corresponding to occurrences $j = i, i+1, \ldots, n-1$ of X. We have three families of rules:

1. **(unary rules).** For each rule $X \overset{\varphi;R}{\Longrightarrow} Y$ in $\mathcal{R}$ and multiset $\beta \in \mathcal{X}^{\oplus}$ of the form $\beta = \gamma + X + \delta$ of size $|\beta| \leq n$, for some $\gamma, \delta \in \mathcal{X}^{\oplus}$, we have a corresponding rule in $\mathcal{R}'$

$$[\beta] \xRightarrow{\varphi|_{X,i};\, R|_{X,i;Y,j};S_{X,i}} [\gamma + Y + \delta]$$

for every occurrence $1 \leq i \leq \beta(X)$ of X in β and for $j = \beta(Y) + 1$, where $\varphi|_{X,i}$ is obtained from φ by replacing each clock x with $x_{X,i}$, and $R|_{X,i;Y,j}$ is obtained from R by replacing every assignment $x := y$ by $x_{Y,j} := x_{X,i}$, and $x := 0$ by $x_{Y,j} := 0$.

2. **(branching rules).** Let $X \Longrightarrow Y + Z$ in $\mathcal{R}$ be a branching rule. We assume w.l.o.g. that it has no tests and no assignments, and that X, Y, Z are pairwise distinct. We add rules in $\mathcal{R}'$

$$[\alpha + X] \xRightarrow{R;S_{X,i}} [\beta], \quad \text{with } \beta = \alpha + Y + Z \text{ and } |\beta| \leq n,$$

for all $1 \leq i \leq \alpha(X)$ and $\alpha \in \mathcal{X}^{\oplus}$, where $R \equiv \bigwedge_{x \in \mathcal{C}} x_{Y,\beta(Y)} := x_{X,i} \wedge x_{Z,\beta(Z)} := x_{X,i}$ copies each clock $x_{X,i}$ into $x_{Y,\beta(Y)}$ and $x_{Z,\beta(Z)}$, and $S_{X,i}$ was defined earlier.

3. **(shrinking rules).** We also add rules that remove unnecessary nonterminals: For every $\beta = \alpha + X \in \mathcal{X}^{\oplus}$ with $|\beta| \leq n$ and index $1 \leq i \leq \beta(X)$ denoting which occurrence of X in β we want to remove, we have a rule $[\alpha + X] \overset{S_i}{\Longrightarrow} [\alpha]$ in $\mathcal{R}'$.

It remains to argue that $(X, \vec{0}) \xrightarrow{*} (T_1, \vec{0}) + \cdots + (T_n, \vec{0})$ in $\mathcal{A}$ if, and only if, $([X], \vec{0}) \xrightarrow{*} ([T_1 + \cdots + T_n], \vec{0})$ in $\mathcal{B}$. This can be proven via induction on the depth of the derivation tree, where the induction hypothesis is that every configuration α of size at most n can be covered in with a derivation tree of depth d in $\mathcal{A}$ if, and only if, in the timed automaton $\mathcal{B}$ the configuration $[\alpha]$ can be reached via a path of length at most d. ◀

▶ **Theorem 13.** *The reachability problem for TBPP is* EXPTIME*-complete. Moreover,* EXPTIME*-hardness already holds for k-TBPP emptiness, if 1) $k \geq 2$ is any fixed number of clocks, or 2) k is part of the input but only* 0 *or* 1 *appear as constants in clock constraints.*

In the remainder of this section contains a proof of this result, in three steps: In the first step (Lemma 14) we show an EXPTIME upper bound for the special case of *simple reachability*, i.e., when the target configuration has size 1. As a second step (Lemma 15) we reduce general case to simple reachability and thereby prove the upper bound claimed in Theorem 13. As a third step (Lemma 16), we prove the corresponding lower bound.

▶ **Lemma 14** (Simple reachability). *The simple reachability problem for TBPP is in* EXPTIME*. More precisely, the complexity is exponential in the number of clocks and the maximal clock constant, and polynomial in the number of nonterminals.*

Proof (Sketch). We reduce to TA reachability games, where two players (Min and Max) alternatingly determine a path of a TA, by letting the player who owns the current nonterminal pick time elapse and a valid successor configuration. Min and Max aim to minimize/maximize the time until the play first visits a target nonterminal T. TA reachability games can be solved in EXPTIME, with the precise time complexity claimed above [30, Theorem 5]. The idea of the construction is to let Min produce a derivation tree along the branch that leads to (unique) target process. Whenever she proposes to go along branching rule, Max gets to claim that the other sibling, not on the main branch, cannot be removed until the main branch ends. This can be faithfully implemented by storing only the current configuration on the main branch plus one more configuration (of Max's choosing) that takes the longest time to vanish. Min can develop both independently but must apply time delays to both simultaneously. Min wins the game if she can reach the target nonterminal and before that moment all the other branches have vanished. ◀

▶ **Lemma 15.** *The reachability problem for TBPP is in* EXPTIME*.*

Proof. First notice that the special case of reachability of the empty target set trivially reduces to the simple reachability problem by adding a dummy nonterminal, which is created once at the beginning and has to be the only one left at the end. Suppose we have an instance of the k-TBPP reachability problem with target nonterminals $T_1, T_2, \ldots, T_m$. We will create an instance of simple reachability where the number of nonterminals increases exponentially but the number of clocks is $O(k \cdot |\mathcal{X}| \cdot m)$. In both cases, the claim follows from Lemma 14.

We introduce a nonterminal $[\beta]$ for every multiset $\beta \in \mathcal{X}^{\oplus}$ of size $|\beta| \leq n := m + 2$, and we have the same three family of rules as in proof of Theorem 12, where the last family 3. is replaced by the family below:

3'. We add extra branching rules in order to maintain nonterminals $[\beta]$ corresponding to small multisets $|\beta| \leq n$. Let $\beta \in \mathcal{X}^{\oplus}$ of size $|\beta| \leq n$ and consider a partitioning $\beta = \beta_1 + \beta_2$, for some $\beta_1, \beta_2 \in \mathcal{X}^{\oplus}$. We identify β with the set $\beta = \{(X, i) \mid X \in \mathcal{X}, 1 \leq i \leq \beta(X)\}$ of pairs (X, i), where i denotes the i-th occurrence of X in β (if any), and similarly for β_1, β_2. We add a branching rule

$$[\beta] \Longrightarrow (\beta, f, \beta_1) + (\beta, f, \beta_2),$$

where (β, f, β_i) are intermediate locations, for every bijection $f : \beta \to \beta_1 \cup \beta_2$ assigning an occurrence of X in β to an occurrence of X either in β_1 or β_2. We then have clock reassigning (non-branching) rules

$$(\beta, f, \beta_1) \overset{S_1}{\Longrightarrow} [\beta_1] \quad \text{and} \quad (\beta, f, \beta_2) \overset{S_2}{\Longrightarrow} [\beta_2],$$

where $S_1 \equiv \bigwedge_{x \in C} \bigwedge_{X \in \mathcal{X}} \bigwedge_{1 \leq i \leq \beta_1(X)} x_{X,i} := x_{f^{-1}(X,i)}$ and similarly for S_2. ◀

▶ **Lemma 16.** *The non-emptiness problem for TBPP is* EXPTIME*-hard already in discrete time, for 1) TBPP with constants in* $\{0, 1\}$ *(where the number of clocks is part of the input), and 2) for* k*-TBPP for every fixed number of clocks* $k \geq 2$*.*

6 Conclusion

We introduced basic parallel processes extended with global time and studied the complexities of several natural decision problems, including variants of the coverability and reachability problems. Table 1 summarizes our findings.

The exact complexity status of the simple reachability problem for 1-TBPP is left open. An NP upper bound holds from the (general) reachability problem (by Theorem 11) and PTIME-hardness comes from the emptiness problem for context-free grammars. We conjecture that a matching polynomial-time upper bound holds.

Also left open for future work are *succinct* versions the coverability and reachability problems, where the target size is given in binary. A reduction from subset-sum games [25] shows that the succinct coverability problem for 1-TBPP is PSPACE-hard. This implies that our technique showing the NP-membership for the non-succinct version of the coverability problem (cf. Theorem 7) does not extend to the succinct variant, and new ideas are needed.

Table 1 Results on TBPP and 1-clock TBPP. The decision problems are complete for the stated complexity class. Simple Coverability/Reachability refer to the variants where the target has size 1.

	Emptiness	Simple Coverability	Coverability	Simple Reachability	Reachability
TBPP	EXPTIME [Lem 16], [30]	PSPACE [Thm 12]	PSPACE [Thm 12]	EXPTIME [Thm 14]	EXPTIME [Thm 13]
1-TBPP	PTIME [33]	NL [33]	NP [Thm 7]	PTIME / NP	NP [Thm 11]

References

1 P. A. Abdulla, M. F. Atig, and J. Cederberg. Timed Lossy Channel Systems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2012. doi:10.4230/LIPIcs.FSTTCS.2012.374.

2 P. A. Abdulla, M. F. Atig, and J. Stenman. Dense-Timed Pushdown Automata. In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2012. doi:10.1109/LICS.2012.15.

3 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Radu Ciobanu, Richard Mayr, and Patrick Totzke. Universal Safety for Timed Petri Nets is PSPACE-complete. In *International Conference on Concurrency Theory (CONCUR)*, 2018. doi:10.4230/LIPIcs.CONCUR.2018.6.

4 Parosh Aziz Abdulla, Mohamed Faouzi Atig, and Shankara Narayanan Krishna. Perfect Timed Communication Is Hard. In *International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS)*, 2018.

5 Parosh Aziz Abdulla and Bengt Jonsson. Model checking of systems with many identical timed processes. *Theoretical Computer Science*, 2003. doi:10.1016/S0304-3975(01)00330-9.

6 Parosh Aziz Abdulla and Aletta Nylén. Timed Petri Nets and BQOs. In *Applications and Theory of Petri Nets and Concurrency (PETRI NETS)*, 2001.

7 Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 1994. doi:10.1016/0304-3975(94)90010-8.

8 M. Benerecetti, S. Minopoli, and A. Peron. Analysis of Timed Recursive State Machines. In *International Symposium on Temporal Representation and Reasoning (TIME)*, 2010. doi:10.1109/TIME.2010.10.

9 Massimo Benerecetti and Adriano Peron. Timed recursive state machines: Expressiveness and complexity. *Theoretical Computer Science*, 2016. doi:10.1016/j.tcs.2016.02.021.

10 Beatrice Bérard, Anne Labroue, and Philippe Schnoebelen. Verifying Performance Equivalence for Timed Basic Parallel Processes. In *International Conference on Foundations of Software Science and Computational Structures (FoSSaCS)*, 2000.

11 Bernard Boigelot, Sébastien Jodogne, and Pierre Wolper. An Effective Decision Procedure for Linear Arithmetic over the Integers and Reals. *ACM Trans. Comput. Logic*, 2005. doi:10.1145/1071596.1071601.

12 Ahmed Bouajjani, Rachid Echahed, and Riadh Robbana. On the Automatic Verification of Systems with Continuous Variables and Unbounded Discrete Data Structures. In *Hybrid Systems (HS)*, 1994.

13 Patricia Bouyer, Kim G. Larsen, Nicolas Markey, and Jacob Illum Rasmussen. Almost Optimal Strategies in One Clock Priced Timed Games. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2006.

14 Søren Christensen. *Decidability and Decomposition in Process Algebras.* PhD thesis, School of Informatics, University of Edinburgh, 1993.

15 Lorenzo Clemente. Decidability of Timed Communicating Automata. *arXiv e-prints*, 2018. `arXiv:1804.07815`.

16 Lorenzo Clemente, Frédéric Herbreteau, Amelie Stainer, and Grégoire Sutre. Reachability of Communicating Timed Processes. In *International Conference on Foundations of Software Science and Computational Structures (FoSSaCS)*, 2013. `doi:10.1007/978-3-642-37075-5_6`.

17 Lorenzo Clemente, Piotr Hofman, and Patrick Totzke. Timed Basic Parallel Processes. *arXiv e-prints*, July 2019. `arXiv:1907.01240`.

18 Lorenzo Clemente and Sławomir Lasota. Binary reachability of timed pushdown automata via quantifier elimination. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2018. `doi:10.4230/LIPIcs.ICALP.2018.118`.

19 Lorenzo Clemente, Sławomir Lasota, Ranko Lazić, and Filip Mazowiecki. Timed pushdown automata and branching vector addition systems. In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2017.

20 Hubert Comon and Yan Jurski. Timed Automata and the Theory of Real Numbers. In *International Conference on Concurrency Theory (CONCUR)*, 1999.

21 Zhe Dang. Binary Reachability Analysis of Pushdown Timed Automata with Dense Clocks. In *Computer Aided Verification (CAV)*, 2001.

22 C. Dima. Computing reachability relations in timed automata. In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2002. `doi:10.1109/LICS.2002.1029827`.

23 Catalin Dima. A Class of Automata for Computing Reachability Relations in Timed Systems. In *Verification of Infinite State Systems with Applications to Security (VISSAS)*, 2005.

24 Javier Esparza. Petri Nets, Commutative Context-Free Grammars, and Basic Parallel Processes. *Fundamenta Informaticae*, 1997.

25 John Fearnley and Marcin Jurdziński. Reachability in two-clock timed automata is PSPACE-complete. *Information and Computation*, 2015. `doi:10.1016/j.ic.2014.12.004`.

26 Jeanne Ferrante and Charles Rackoff. A Decision Procedure for the First Order Theory of Real Addition with Order. *SIAM Journal on Computing*, 1975.

27 Martin Fränzle, Karin Quaas, Mahsa Shirmohammadi, and James Worrell. Effective Definability of the Reachability Relation in Timed Automata. *arXiv e-prints*, 2019. `arXiv:1903.09773`.

28 Serge Haddad, Sylvain Schmitz, and Philippe Schnoebelen. The Ordinal Recursive Complexity of Timed-Arc Petri Nets, Data Nets, and Other Enriched Nets. In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2012. `doi:10.1109/LICS.2012.46`.

29 Thomas Dueholm Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen. A Faster Algorithm for Solving One-Clock Priced Timed Games. In *International Conference on Concurrency Theory (CONCUR)*, 2013.

30 Marcin Jurdziński and Ashutosh Trivedi. Reachability-time Games on Timed Automata. In *International Colloquium on Automata, Languages and Programming (ICALP)*, 2007.

31 Pavel Krčál and Radek Pelánek. On Sampled Semantics of Timed Systems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2005.

32 Pavel Krcal and Wang Yi. Communicating timed automata: the more synchronous, the more difficult to verify. In *Computer Aided Verification (CAV)*, 2006. `doi:10.1007/11817963_24`.

33 F. Laroussinie, N. Markey, and Ph. Schnoebelen. Model Checking Timed Automata with One or Two Clocks. In *International Conference on Concurrency Theory (CONCUR)*, 2004.

34 Kim G. Larsen, Paul Pettersson, and Wang Yi. Uppaal in a Nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1997. doi:10.1007/s100090050010.

35 Slawomir Lasota and Igor Walukiewicz. Alternating timed automata. *ACM Trans. Comput. Logic*, 2008. doi:10.1145/1342991.1342994.

36 Philip Meir Merlin. *A Study of the Recoverability of Computing Systems*. PhD thesis, University of California, Irvine, 1974.

37 J. Ouaknine and J. Worrell. On the decidability of metric temporal logic. In *ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 2005. doi:10.1109/LICS.2005.33.

38 Louchka Popova. On Time Petri Nets. *Journal of Information Processing and Cybernetics*, 1991.

39 Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. *Comptes Rendus Premier Congrès des Mathématicienes des Pays Slaves*, 1930.

40 Karin Quaas. Verification for Timed Automata extended with Unbounded Discrete Data Structures. *Logical Methods in Computer Science*, 2015. doi:10.2168/LMCS-11(3:20)2015.

41 Valentin Valero Ruiz, Fernando Cuartero Gomez, and David de Frutos Escrig. On Non-Decidability of Reachability for Timed-Arc Petri Nets. In *International Workshop on Petri Nets and Performance Models (PNPM)*, 1999.

42 Ashutosh Trivedi. *Competative Optimisation on Timed Automata*. PhD thesis, University of Warwick, 2009.

43 Ashutosh Trivedi and Dominik Wojtczak. Recursive Timed Automata. In *International Symposium on Automated Technology for Verification and Analysis (ATVA)*, 2010.

44 Kumar Neeraj Verma, Helmut Seidl, and Thomas Schwentick. On the Complexity of Equational Horn Clauses. In *International Conference on Automated Deduction (CADE)*, 2005. doi:10.1007/11532231_25.

45 Volker Weispfenning. Mixed Real-integer Linear Quantifier Elimination. In *International Symposium on Symbolic and Algebraic Computation (ISSAC)*, 1999. doi:10.1145/309831.309888.

46 Sergio Yovine. KRONOS: a verification tool for real-time systems. *International Journal on Software Tools for Technology Transfer*, 1997. doi:10.1007/s100090050009.

Revisiting Local Time Semantics for Networks of Timed Automata

R. Govind
Chennai Mathematical Institute, India
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400, Talence, France
govindr@cmi.ac.in

Frédéric Herbreteau
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400, Talence, France
fh@labri.fr

B. Srivathsan
Chennai Mathematical Institute, India
sri@cmi.ac.in

Igor Walukiewicz
Univ. Bordeaux, CNRS, Bordeaux INP, LaBRI, UMR 5800, F-33400, Talence, France
igw@labri.fr

Abstract

We investigate a zone based approach for the reachability problem in timed automata. The challenge is to alleviate the size explosion of the search space when considering networks of timed automata working in parallel. In the timed setting this explosion is particularly visible as even different interleavings of local actions of processes may lead to different zones. Salah *et al.* in 2006 have shown that the union of all these different zones is also a zone. This observation was used in an algorithm which from time to time detects and aggregates these zones into a single zone.

We show that such aggregated zones can be calculated more efficiently using the local time semantics and the related notion of local zones proposed by Bengtsson *et al.* in 1998. Next, we point out a flaw in the existing method to ensure termination of the local zone graph computation. We fix this with a new algorithm that builds the local zone graph and uses abstraction techniques over (standard) zones for termination. We evaluate our algorithm on standard examples. On various examples, we observe an order of magnitude decrease in the search space. On the other examples, the algorithm performs like the standard zone algorithm.

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

Keywords and phrases Timed automata, verification, local-time semantics, abstraction

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.16

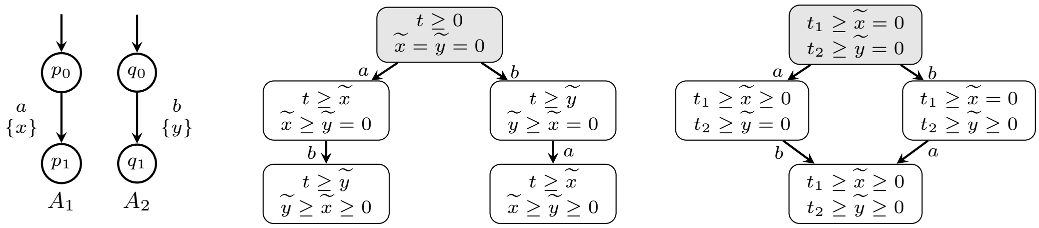
Related Version A full version of this paper is available at [10], `http://arxiv.org/abs/1907.02296`.

Funding Work supported by UMI 2000 ReLaX, ANR project TickTac, and project IoTTTA – CEFIPRA Indo-French program in ICST – DST/CNRS ref. 2016-01. Author B. Srivathsan is partially funded by grants from Infosys Foundation, India and Tata Consultancy Services, India.

1 Introduction

Timed automata [1] are a popular model for real-time systems. They extend finite state automata with real valued variables called *clocks*. Constraints on clock values can be used as guards for transitions, and clocks can be reset to zero during transitions. Often, it is more natural to use a network of timed automata which operate concurrently and synchronize on joint actions. We study the reachability problem for networks of timed automata: given a state of the timed automaton network, is there a run from the initial state to the given state.

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 16; pp. 16:1–16:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany


p_0
q_0
a
$\{x\}$
b
$\{y\}$
p_1
q_1
A_1
A_2
$t \geq 0$
$\widetilde{x} = \widetilde{y} = 0$
a
b
$t \geq \widetilde{x}$
$\widetilde{x} \geq \widetilde{y} = 0$
$t \geq \widetilde{y}$
$\widetilde{y} \geq \widetilde{x} = 0$
b
a
$t \geq \widetilde{y}$
$\widetilde{y} \geq \widetilde{x} \geq 0$
$t \geq \widetilde{x}$
$\widetilde{x} \geq \widetilde{y} \geq 0$
$t_1 \geq \widetilde{x} = 0$
$t_2 \geq \widetilde{y} = 0$
a
b
$t_1 \geq \widetilde{x} \geq 0$
$t_2 \geq \widetilde{y} = 0$
$t_1 \geq \widetilde{x} = 0$
$t_2 \geq \widetilde{y} \geq 0$
b
a
$t_1 \geq \widetilde{x} \geq 0$
$t_2 \geq \widetilde{y} \geq 0$

Network $\langle A_1, A_2 \rangle$ Global zone graph Local zone graph

Figure 1 Illustration of commutativity in the local zone graph.

A widely used technique to solve the reachability problem constructs a *zone graph* [7] whose nodes are *(state, zone)* pairs consisting of a state of the automaton and a *zone* representing a set of clock valuations [8]. This graph may not be finite, so in order to guarantee termination of an exploration algorithm, various sound and complete abstraction techniques are used [7, 2, 12, 9].

Dealing with automata operating in parallel poses the usual state-space explosion problem arising due to different interleavings. Consider an example of a network with two processes in Figure 1. Actions a and b are local to each process. Variables x, y are clocks and $\{x\}$ denotes that clock x is reset in transition a. The (global) zone graph maintains the set of configurations reached after each sequence of actions. Although a and b are local actions, there is an intrinsic dependence between the two processes happening due to time. Hence, the zone reached after executing sequence ab contains configurations where x is reset before y and the zone after ba contains configurations where y is reset before x. So the sequences ab and ba lead to different zones. The number of different interleavings of sequences of local actions increases exponentially when their length grows, or when more processes get involved. Every interleaving can potentially lead to a different zone.

Salah et al. [17] have shown a surprising property that the *union* of all zones reached by the interleavings of a sequence of actions is also a zone. We call it an *aggregated zone*. Their argument is based on the fact that for a given sequence one can write a zone-like constraint defining all the runs on the interleavings of the sequence. Then the aggregated zone is obtained simply by projecting this big constraint on relevant components. This approach requires to work with sets of constraints whose size grows with the length of a sequence. This is both inefficient and limited to finite sequences. They use this observation in an algorithm where, when all the interleavings of a sequence σ have been explored, the resulting zones are aggregated to a single zone and further exploration is restricted to this aggregated zone. This requires detecting from time to time whether aggregation can happen. This is an obstacle in using aggregated zones in efficient reachability-checking algorithms. Another limitation of this approach is that it works only for acyclic automata.

Another approach by Bengtsson et al. [4] involves making time local to each process. This local time approach is based on a very elegant idea: make time in every process progress independently, and synchronize local times of processes when they need to perform a common action. In consequence, the semantics has the desired property: two actions whose process domains are disjoint are commutative. In the example above, depending on the local time in processes A_1 and A_2, the sequence ab may result, on a global time scale, in a occurring before b, as well as b occurring before a. As a result, the set of valuations reached after ab does not remember the order in which a and b occurred. Thus, sequences ab and ba lead to the same set of configurations: those obtained after doing a and b concurrently. Similar to standard zones, we now have local zones and a local zone graph having *(state, local zone)* pairs. Due to the above argument, the local zone graph has a nice property – if a run σ from an initial zone reaches a local zone, then all the runs equivalent to σ lead to the same local zone. Local zone graphs are therefore ideal for handling interleavings. However, substantially more involved abstraction techniques are needed for local zones to make the local zone graph finite. A *subsumption relation* between local zones was defined in [4] but no algorithm was proposed and there was no effective way to use local time semantics. Later Minea [15] proposed a widening operator on local zones to construct finite local zone graphs.

Summary of results in the paper:

- We show that the aggregated zone of interleavings of σ in the standard zone graph is obtained by synchronizing valuations in the local zone obtained after σ (Theorem 18). Hence computing the local zone graph gives a more direct and efficient algorithm than Salah et al. to compute aggregated zones. Moreover, this algorithm is not restricted to acyclic timed automata.
- We point out a flaw in the abstraction procedure of Minea to get a finite local zone graph (Section 5).
- We propose a different algorithm to get a finite local zone graph. This gives a new reachability algorithm for networks of timed automata, which works with local zones but uses subsumption on standard zones (Definition 19 and Theorem 20). Instead of subsumptions between local zones, we use subsumptions between the synchronized valuations inside these local zones. This helps us to exploit the (well-studied) subsumptions over standard zones [7, 2, 12, 9]. Moreover, this subsumption is much more aggressive than the standard one since, thanks to local-time semantics, the (aggregated) zone used in the subsumption represents all the valuations reachable not only by the execution that we are exploring but also by all the executions equivalent to it.
- We report on experiments performed with a prototype implementation of the algorithm (Table 1). The algorithm performs surprisingly well on some examples, and it is never worse than the standard zone graph algorithm.

Related work. The basis of this work is local-time semantics and local zones developed by Bengtsson et. al. [4]. The authors have left open how to use this semantics to effectively compute the local zone graph since no efficient procedure was provided to ensure its finiteness. To that purpose, Minea [16, 15] has proposed a subsumption operation on local zones, and an algorithm using this operation. Unfortunately, as we exhibit here, the algorithm has a flaw that is not evident to repair. Lugiez et. al. [14] also use local time but their method is different. They use constraints to check if local clocks of an execution can be synchronized sufficiently to obtain a standard run.

Aggregated zones are crucial to obtain an efficient verification procedure for networks of timed automata. Coming to the same state with different zones inflicts a huge blowup in the zone graph, since the same paths are explored independently from each of these zones. This has also been observed in the context of multi-threaded program verification in [18]. Solving this problem in the context of program analysis requires to over approximate the aggregated state. Fortunately, in the context of timed automata, the result of aggregating these zones in still a zone [17]. Efficient computation of aggregated zones is thus an important advance in timed automata verification as demonstrated by our experimental results in Section 6. A full version of this paper is available at [10].

2 Networks of timed automata

We start by defining networks of timed automata and two semantics for them: a global-time semantics (the usual one) and a local-time semantics (introduced in [4]). Then, we recall the fact that they are equivalent, and state some interesting properties of local-time semantics w.r.t. concurrency that were observed in [4]. Let $\mathbb{N}$ denote the set of natural numbers and $\mathbb{R}_{\geq 0}$ the set of non-negative reals. Let X be a finite set of variables called *clocks*. Let $\phi(X)$ denote a set of clock constraints generated by the following grammar: $\phi := x \sim c \mid \phi \wedge \phi$ where $x \in X$, $c \in \mathbb{N}$, and $\sim \in \{<, \leq, =, \geq, >\}$.

▶ **Definition 1** (Network of timed automata). *A network of timed automata with k processes, is a k-tuple of timed automata $A_1, \ldots, A_k$. Each process $A_p = \langle Q_p, \Sigma_p, X_p, q_p^{init}, T_p \rangle$ has a finite set of states Q_p, a finite alphabet of actions Σ_p, a finite set of clocks X_p, an initial state q_p^{init}, and transitions $T_p \subseteq \Sigma_p \times Q_p \times \phi(X_p) \times 2^{X_p} \times Q_p$. We require that the sets of states, and the sets of clocks are pairwise disjoint: $Q_{p_1} \cap Q_{p_2} = \emptyset$, and $X_{p_1} \cap X_{p_2} = \emptyset$ for $p_1 \neq p_2$. We write Proc for the set of all processes.*

A network is a parallel composition of timed automata. Its semantics is that of the timed automaton obtained as the "synchronized product" of the processes. For an action b, a b-transition of a process p is an element of T_p with b in the first component. Synchronization happens on two levels: (i) via time that advances the same way in all the processes, and (ii) via common actions, for example if $b \in \Sigma_1 \cap \Sigma_2$, then processes 1 and 2 need to synchronize by doing a b-transition. We define the domain of an action b: $dom(b) = \{p : b \in \Sigma_p\}$ as the set of processes that must synchronize to do b. We will use some abbreviations: $Q = \Pi_{p=1}^{k} Q_p$, $\Sigma = \bigcup_{p=1}^{k} \Sigma_p$ and $X = \bigcup_{p=1}^{k} X_p$.

The semantics of a network is governed by the value of clocks at each instant. We choose to represent these values using offsets as this allows a uniform presentation of the global-time semantics below and the local-time semantics in Section 2.1. For every clock x, we introduce an offset variable $\widetilde{x}$. The value of $\widetilde{x}$ is the time-stamp at which x was last reset. In addition, we consider a variable t which tracks the global time: essentially t is a clock that is never reset. We now make this notion precise. Let $\widetilde{X}_p = \{\widetilde{x} \mid x \in X_p\}$ and $\widetilde{X} = \bigcup_{p=1}^{k} \widetilde{X}_p$. A *global valuation* v is a function $v : \widetilde{X} \cup \{t\} \mapsto \mathbb{R}_{\geq 0}$ such that $v(t) \geq v(\widetilde{x})$ for all variables $\widetilde{x}$. In this representation, the value of clock x corresponds to $v(t) - v(\widetilde{x})$, denoted $\overline{v}(x)$[1].

Recall that offset variable $\widetilde{x}$ stores the last time-stamp at which clock x has been reset. Hence, a delay in offset representation increases the value of reference clock t and leaves offset variables $\widetilde{x}$ unchanged. Formally: for $\delta \in \mathbb{R}_{\geq 0}$, we denote by $v + \delta$ the valuation defined by: $(v + \delta)(t) = v(t) + \delta$ and $(v + \delta)(\widetilde{x}) = v(\widetilde{x})$ for all $\widetilde{x}$. Similarly, resetting the clocks in $R \subseteq X$, yields a global valuation $[R]v$ defined by: $([R]v)(t) = v(t)$, and $([R]v)(\widetilde{x})$ is $v(t)$ if $x \in R$, and $v(\widetilde{x})$ otherwise. Given a global valuation v and a clock constraint g, we write $v \models g$ if every constraint in g holds after replacing x with its value $\overline{v}(x) = v(t) - v(\widetilde{x})$.

A configuration of the network is a pair (q, v) where $q \in Q$ is a global state, and v is a global valuation. We will write $q(p)$ to refer to the p-th component of the state q.

▶ **Definition 2** (Global-time semantics). *The semantics of a network $\mathcal{N}$ is given by a transition system whose states are configurations (q, v). The initial configuration is (q^{init}, v^{init}) where $q^{init}(p) = q_p^{init}$ is the tuple of initial states, and $v^{init}(y) = 0$ for $y \in \widetilde{X} \cup \{t\}$.*

[1] Usually, semantics of timed automata is described using valuations of the form $\overline{v}$. Here, we have chosen v which uses offsets since it extends naturally to the local-time setting. Translations between these two kinds of valuations is straightforward, as shown.

There are two kinds of transitions, which we call steps: global delay, and action steps. A global delay *by the amount $\delta \in \mathbb{R}_{\geq 0}$ gives a step $(q, v) \xrightarrow{\delta}_{st} (q, v + \delta)$. An* action step *on action b gives $(q, v) \xrightarrow{b}_{st} (q', v')$ if there is a set of b-transitions $\{(q_p, g_p, R_p, q'_p)\}_{p \in dom(b)}$ of the respective processes such that:*

- *processes from $dom(b)$ change states: $q_p = q(p)$, $q'_p = q'(p)$, for $p \in dom(b)$, and $q(p) = q'(p)$ for $p \notin dom(b)$;*
- *all the guards are satisfied: $v \vDash g_p$, for $p \in dom(b)$;*
- *all resets are performed: $v' = [\bigcup_{p \in dom(b)} R_p]v$;*

A *run of an automaton* from a configuration (q_0, v_0) is a sequence of steps starting in (q_0, v_0). For a sequence $u = b_1 \dots b_n$ of actions, we write $(q_0, v_0) \xRightarrow{u} (q_n, v'_n)$ if there is a run

$$(q_0, v_0) \xrightarrow{\delta_0}_{st} (q_0, v'_0) \xrightarrow{b_1}_{st} (q_1, v_1) \xrightarrow{\delta_1}_{st} (q_1, v'_1) \dots \xrightarrow{b_n}_{st} (q_n, v_n) \xrightarrow{\delta_n}_{st} (q_n, v'_n)$$

for some delays $\delta_0, \dots, \delta_n \in \mathbb{R}_{\geq 0}$.

▶ **Definition 3** (Reachability problem). *The* reachability problem *is to decide, given a network $\mathcal{N}$ and a state q, whether there is a run reaching q; or in other words, whether there exists a sequence of transitions u such that $(q^{init}, v^{init}) \xRightarrow{u} (q, v)$ for some valuation v.*

The reachability problem for networks of timed automata is PSPACE-complete [1].

2.1 Local-time semantics

The definition of a network of automata suggests an independence relation between actions: a pair of actions with disjoint domains (i.e. involving distinct processes) should commute. We say that two sequences of actions are *equivalent*, written $u \sim w$ if one can be obtained from the other by repeatedly permuting adjacent actions with disjoint domains.

▶ **Lemma 4.** *For two equivalent sequences $u \sim w$: if there are two runs $(q, v) \xRightarrow{u} (q_u, v_u)$, and $(q, v) \xRightarrow{w} (q_w, v_w)$ then $q_u = q_w$.*

Observe that in the above lemma we cannot assert that $v_u = v_w$. Even further, the existence of $(q, v) \xRightarrow{u} (q_u, v_u)$ does not imply that a run from (q, v) on w is feasible. This happens due to global time delays, i.e., delays that involve all the processes. For example, consider actions a and b on disjoint processes with a having guard $x \leq 1$ and b having guard $y \geq 2$. Then from the initial valuation one can execute ab but not ba.

One solution to get commutativity between actions with disjoint domains is to consider local-time semantics [4]. In this semantics, time elapses independently in every process, and time elapse is synchronized before executing a synchronized action. This way, two actions with disjoint domains become commutative. In the example from the previous paragraph, while the process executing b elapses 2 time units, the other process is allowed to not elapse time at all and hence ba becomes possible. Moreover, for the reachability problem, local-time semantics is equivalent to the standard one.

In the local-time semantics, we replace the clock t which was tracking the global time, with individual *reference clocks* t_p for each process A_p which track the local time of each process. We set $\widetilde{X}'_p = \widetilde{X}_p \cup \{t_p\}$ and $\widetilde{X}' = \bigcup_p \widetilde{X}'_p$. A *local valuation* v is a valuation over the set of clocks $\widetilde{X}'$ such that $\mathsf{v}(t_p) \geq \mathsf{v}(\widetilde{x})$ for all processes $p \in Proc$ and all clocks $\widetilde{x} \in \widetilde{X}_p$. This restriction captures the intuition that t_p is a reference clock for process p, and it is never reset. In this setting, the value $\mathsf{v}(t_p) - \mathsf{v}(\widetilde{x})$ of clock x is defined relative to the reference clock t_p of process p that owns x, i.e. $x \in X_p$. We will use the notation v for local valuations to distinguish from global valuations v.

We introduce a new operation of local time elapse. For a process $p \in \mathit{Proc}$ and $\delta \in \mathbb{R}_{\geq 0}$, operation $\mathsf{v} +_p \delta$ adds δ to $\mathsf{v}(t_p)$, the value of the reference clock t_p of process p, and leaves the other variables unchanged. Formally, $(\mathsf{v} +_p \delta)(t_p) = \mathsf{v}(t_p) + \delta$ and $(\mathsf{v} +_p \delta)(y) = \mathsf{v}(y)$ for all $y \in \widetilde{X}' \setminus \{t_p\}$. A local valuation v satisfies a clock constraint g, denoted $\mathsf{v} \models g$ if every constraint in g holds after replacing x by its value $\mathsf{v}(t_p) - \mathsf{v}(\widetilde{x})$ where p is the process such that $x \in X_p$. We denote by $[R]\mathsf{v}$ the valuation obtained after resetting the clocks in $R \subseteq X$ and defined by: $([R]\mathsf{v})(t_p) = \mathsf{v}(t_p)$ for every reference clock t_p, $([R]\mathsf{v})(\widetilde{x}) = \mathsf{v}(\widetilde{x})$ if $x \notin R$, and $([R]\mathsf{v})(\widetilde{x}) = \mathsf{v}(t_p)$ if $x \in R$ and p is the process such that $x \in X_p$.

▶ **Definition 5** (Local steps of a timed automata network). *There are two kinds of local steps in a network $\mathcal{N}$:* local delay*, and* local action*. A local delay $\delta \in \mathbb{R}_{\geq 0}$ in process $p \in \mathit{Proc}$ is a step $(q, \mathsf{v}) \xrightarrow{p,\delta}_{st} (q, \mathsf{v} +_p \delta)$. For an action b, we have a step $(q, \mathsf{v}) \xrightarrow{b}_{st} (q', \mathsf{v}')$ if there is a set of b-transitions of respective processes $\{(q_p, g_p, R_p, q'_p)\}_{p \in dom(b)}$ such that:*

- *$q_p = q(p)$, $q'_p = q'(p)$, for $p \in dom(b)$, and $q(p) = q'(p)$ for $p \notin dom(b)$;*
- *start times are synchronized: $\mathsf{v}(t_{p_1}) = \mathsf{v}(t_{p_2})$, for every $p_1, p_2 \in dom(b)$;*
- *guards are satisfied: $\mathsf{v} \vDash g_p$, for every $p \in dom(b)$;*
- *resets are performed: $\mathsf{v}' = [\bigcup_{p \in dom(b)} R_p]\mathsf{v}$;*

The main difference with respect to global semantics is the presence of local time delay. As a result, every process can be in a different local time as emphasized by the reference clocks in each process. In consequence, in local action steps we require that when processes do a common action their local times should be the same. Of course a standard delay δ on all processes can be simulated by a sequence of delays on every process separately, as $\xrightarrow{1,\delta}_{st} \cdots \xrightarrow{k,\delta}_{st}$. For a sequence of local delays $\Delta = (p_1, \delta_1) \dots (p_n, \delta_n)$ we will write $(q, \mathsf{v}) \xrightarrow{\Delta}_{st} (q, \mathsf{v}')$ to mean $(q, \mathsf{v}) \xrightarrow{p_1,\delta_1}_{st} (q, \mathsf{v}_1) \xrightarrow{p_2,\delta_2}_{st} \cdots \xrightarrow{(p_n,\delta_n)}_{st} (q, \mathsf{v}')$.

▶ **Definition 6** (Local run). *A* local run *from a configuration (q_0, v_0) is a sequence of local steps. For a sequence of actions $u = b_1 \dots b_n$, we write $(q_0, \mathsf{v}_0) \xrightarrow{u} (q_n, \mathsf{v}'_n)$ if for some sequences of local delays $\Delta_0, \dots, \Delta_n$ there is a local run*

$$(q_0, \mathsf{v}_0) \xrightarrow{\Delta_0}_{st} (q_0, \mathsf{v}'_0) \xrightarrow{b_1}_{st} (q_1, \mathsf{v}_1) \xrightarrow{\Delta_1}_{st} \cdots \xrightarrow{b_n}_{st} (q_n, \mathsf{v}_n) \xrightarrow{\Delta_n}_{st} (q_n, \mathsf{v}'_n)$$

Observe that a run may start and end with a sequence of delays. In the next section we will make a link between local and global runs. For this we will first examine independence properties of local runs which are much better than for global runs (cf. Lemma 4).

▶ **Lemma 7** (Independence). *Suppose $dom(a) \cap dom(b) = \emptyset$. If $(q, \mathsf{v}) \xrightarrow{ab} (q', \mathsf{v}')$ then $(q, \mathsf{v}) \xrightarrow{ba} (q', \mathsf{v}')$. If $(q, \mathsf{v}) \xrightarrow{a} (q_a, \mathsf{v}_a)$ and $(q, \mathsf{v}) \xrightarrow{b} (q_b, \mathsf{v}_b)$ then $(q, \mathsf{v}) \xrightarrow{ab} (q_{ab}, \mathsf{v}_{ab})$ for some q_{ab} and v_{ab}.*

Recall that two sequences of actions are equivalent, written $u \sim w$ if one can be obtained from the other by repeatedly permuting adjacent actions with disjoint domains. Directly from the previous lemma we obtain.

▶ **Lemma 8.** *If $(q_0, \mathsf{v}_0) \xrightarrow{u} (q_n, \mathsf{v}_n)$ and $u \sim w$ then $(q_0, \mathsf{v}_0) \xrightarrow{w} (q_n, \mathsf{v}_n)$.*

With local-time semantics two equivalent sequences not only reach the same state q_n, but also the same local valuation v_n (in contrast with Lemma 4 for global-time semantics).

2.2 Comparing local and global runs

We have presented two semantics for networks of timed automata: global-time and local-time. Local runs have more freedom as time can elapse independently in every process. Yet, with respect to state reachability the two concepts turn out to be equivalent.

▶ **Definition 9.** *A local valuation* v *is* synchronized *if for every pair of processes* p_1, p_2*, the values of their reference clocks are equal:* $\mathsf{v}(t_{p_1}) = \mathsf{v}(t_{p_2})$*.*

For a synchronized local valuation v, let $\mathsf{global}(\mathsf{v})$ be the global valuation v such that $v(\widetilde{x}) = \mathsf{v}(\widetilde{x})$ and $v(t) = \mathsf{v}(t_1) = \cdots = \mathsf{v}(t_k)$. Conversely, to every global valuation v, we associate the synchronized local valuation $\mathsf{local}(v) = \mathsf{v}$ where $\mathsf{v}(\widetilde{x}) = v(\widetilde{x})$ and $\mathsf{v}(t_p) = v(t)$ for every reference clock t_p.

▶ **Lemma 10.** *If* $(q, \mathsf{v}) \xrightarrow{u} (q', \mathsf{v}')$ *is a local run where* v *and* v' *are synchronized local valuations, there exists a global run* $(q, \mathsf{global}(\mathsf{v})) \xRightarrow{w} (q', \mathsf{global}(\mathsf{v}'))$ *for some* $w \sim u$*. Conversely, if* $(q, v) \xRightarrow{u} (q', v')$ *is a global run, then there is a local run* $(q, \mathsf{local}(v)) \xrightarrow{u} (q', \mathsf{local}(v'))$*.*

The reachability problem with respect to local semantics is defined as before: q is reachable if there is a local run $(q^{init}, \mathsf{v}^{init}) \xrightarrow{u} (q, \mathsf{v})$ for some v where $\mathsf{v}^{init} = \mathsf{local}(v^{init})$. By adding some local delays at the end of the run we can always assume that v is synchronized. Lemma 10 thus implies that the reachability problem in local semantics is equivalent to the standard one in global semantics.

3 Zone graphs

We introduce zones, a standard approach for solving reachability in timed automata. Zones are sets of valuations that can be represented efficiently using simple constraints.

Let us fix a network $\mathcal{N}$ of timed automata with k processes. Recall that each process p has a set of clocks X_p and corresponding offset variables $\widetilde{X}_p$. The set of clocks in $\mathcal{N}$ is $X = \bigcup_{i=1}^{k} X_p$. Similarly, the set of offset variables in $\mathcal{N}$ is $\widetilde{X} = \bigcup_{i=1}^{k} \widetilde{X}_p$.

3.1 Standard zone-based algorithm for reachability

Recall that in the global-time semantics, $\mathcal{N}$ has a reference clock t. A *global zone* is a set of global valuations described by a conjunction of constraints of the form $y_1 - y_2 \lhd c$ where $y_1, y_2 \in \widetilde{X} \cup \{t\}$, $\lhd \; \in \{<, \leq\}$ and $c \in \mathbb{Z}$. Since global valuations need to satisfy $v(\widetilde{x}) \leq v(t)$ for every offset variable $\widetilde{x} \in \widetilde{X}$, a global zone satisfies $\widetilde{x} \leq t$ for every $\widetilde{x} \in \widetilde{X}$.

Let g be a guard and R a set of clocks. We define the following operations on zones: $Z_g = \{v \mid v \models g\}$ is the set of global valuations satisfying g, $[R]Z := \{[R]v \mid v \in Z\}$ and $\overrightarrow{Z} := \{v \mid \exists v' \in Z,\ \exists \delta \in \mathbb{R}_{\geq 0} \text{ s. t. } v = v' + \delta\}$. From [4], Z_g, $[R]Z$ and $\overrightarrow{Z}$ are all zones. We say that a zone is *time-elapsed* if $Z = \overrightarrow{Z}$.

The semantics of a network of timed automata can be described in terms of global zones. For an action b, consider a set of b-transitions of respective processes $\{(q_p, g_p, R_p, q'_p)\}_{p \in dom(b)}$. Let $R = \bigcup_{p \in dom(b)} R_p$, and $g = \bigwedge_{p \in dom(b)} g_p$. Then we have a transition $(q, Z) \xRightarrow{b} (q', Z')$ where $Z' = \overrightarrow{[R](Z \cap Z_g)}$ provided that $q(p) = q_p$, $q'(p) = q'_p$ if $p \in dom(b)$, and $q'(p) = q(p)$ otherwise, and Z' is not empty. We write $\Rightarrow$ for the union over all $\xRightarrow{b}$.

The *global zone graph* $ZG(\mathcal{N})$ of a timed automaton network $\mathcal{N}$ is a transition system whose nodes are of the form (q, Z) where $q \in Q$ and Z is a time-elapsed global zone. The transition relation is given by $\Rightarrow$. The initial node is (q^{init}, Z^{init}) where q^{init} is the tuple of initial states and $Z^{init} = \overrightarrow{\{v^{init}\}}$ where v^{init} is the initial global valuation. The zone graph $ZG(\mathcal{N})$ is known to be sound and complete with respect to reachability. This means that a state q is reachable by a run of $\mathcal{N}$ iff a node (q, Z) for some non-empty Z is reachable from (q^{init}, Z^{init}) in the zone graph. As zone graphs may be infinite, an abstraction operator is used to obtain a finite quotient.

An *abstraction operator* $\mathfrak{a} : P(\mathbb{R}^X_{\geq 0}) \to P(\mathbb{R}^X_{\geq 0})$ [2] is a function from sets of valuations to sets of valuations such that $W \subseteq \mathfrak{a}(W)$ and $\mathfrak{a}(\mathfrak{a}(W)) = \mathfrak{a}(W)$. Simulation relations between valuations are a convenient way to construct abstraction operators that are correct for reachability. A *time-abstract simulation* is a relation between valuations that depends on a given network $\mathcal{N}$. We say that v_1 can be simulated by v_2, denoted $v_1 \preccurlyeq v_2$ if for every state q of $\mathcal{N}$, and every delay-action step $(q, v_1) \overset{\delta_1}{\Longrightarrow}\overset{b}{\Longrightarrow} (q', v_1')$ there is a delay $\delta_2 \in \mathbb{R}_{\geq 0}$ such that $(q, v_2) \overset{\delta_2}{\Longrightarrow}\overset{b}{\Longrightarrow} (q', v_2')$ and $v_1' \preccurlyeq v_2'$. The simulation relation can be lifted to global zones: we say that Z is simulated by Z', written as $Z \preccurlyeq Z'$ if for all $v \in Z$ there exists a $v' \in Z'$ such that $v \preccurlyeq v'$. An abstraction $\mathfrak{a}$ based on $\preccurlyeq$ is defined as $\mathfrak{a}(W) = \{v \mid \exists v' \in W \text{ with } v \preccurlyeq v'\}$. The abstraction $\mathfrak{a}$ is finite if its range is finite. Given two nodes (q, Z) and (q', Z') of $ZG(\mathcal{N})$, (q, Z) is *subsumed* by (q', Z'), denoted $(q, Z) \sqsubseteq^{\mathfrak{a}} (q', Z')$, if $q = q'$ and $Z \subseteq \mathfrak{a}(Z')$.

▶ Remark 11. Our definition of zones slightly differs from the standard definition in the literature (e.g. [5]) since we use offset variables to represent clock valuations. Yet, finite time-abstract simulations from the literature [2, 12] can be adapted to our settings as a simulation over standard valuations $\overline{v}$ can be expressed as a simulation over global valuations v since for every clock x, $\overline{v}(x) = v(t) - v(\widetilde{x})$, and zones over valuations v can be translated to zones over standard valuations $\overline{v}$.

A finite abstraction $\mathfrak{a}$ allows to construct a finite *global zone graph with subsumption* for a network of timed automata $\mathcal{N}$. The construction starts from the initial node of $ZG(\mathcal{N})$. Using, say, a breadth-first-search (BFS), for every constructed node we examine all its successors in $ZG(\mathcal{N})$, and keep only those that are maximal w.r.t. to $\sqsubseteq^{\mathfrak{a}}$ relation. Computing such a zone graph with subsumption gives an algorithm for the reachability problem. However, the global zone graph, and hence the algorithm above, are sensitive to the combinatorial explosion arising from parallel composition. Global time makes any two actions potentially dependent – the same is still true on the level of zones. Zone graphs based on the local-time semantics, as presented next, solve this problem.

3.2 Local zone graphs

The goal of this section is to introduce a concept similar to global zones and global zone graphs for local-time semantics. Recall that in the local-time semantics, each process p has a reference clock t_p.

A *local zone* is a zone over local valuations: a set of local valuations defined by constraints $y_1 - y_2 \lhd c$ where $y_1, y_2 \in \widetilde{X} \cup \{t_1, \dots t_k\}$. Recall that a local valuation v needs to satisfy: $\mathsf{v}(\widetilde{x}) \leq \mathsf{v}(t_p)$ for every process p and every $\widetilde{x} \in \widetilde{X}_p$. This means that a local zone satisfies $\widetilde{x} \leq t_p$ for every process p and every $\widetilde{x} \in \widetilde{X}_p$. We will use Z, eventually with subscripts, to range over local zones, and distinguish from global zones Z. Local zones are closed under all basic operations involved in a local step of a network of timed automata, namely: local time elapse, intersection with a guard, and reset of clocks [4]. That is, for every local zone Z:

- the set local-elapse$(\mathsf{Z}) = \{\mathsf{v} +_1 \delta_1 +_2 \cdots +_k \delta_k \mid \mathsf{v} \in \mathsf{Z},\ \delta_1, \dots, \delta_k \in \mathbb{R}_{\geq 0}\}$ is a local zone.
- for every guard g the set $\mathsf{Z}_g = \{\mathsf{v} \mid \mathsf{v} \vDash g\}$ is a local zone.
- for every set of clocks $R \subseteq X$, the set $[R]\mathsf{Z} = \{[R]\mathsf{v} \mid \mathsf{v} \in \mathsf{Z}\}$ is a local zone.

Local zones can be implemented using DBMs. Hence, they can be computed and stored as efficiently as standard zones.

The operations of local time elapse, guard intersection, and reset, enable us to describe a local step $(q, \mathsf{Z}) \xrightarrow{b} (q', \mathsf{Z}')$ on the level of local zones. This is done in the same way as for global zones. Observe that a local step is indexed only by an action, as the time is taken care of by the local time elapse operation. Formally, for an action b consider a set of b-transitions $\{(q_p, g_p, R_p, q'_p)\}_{p \in dom(b)}$ of respective processes. Then we have a transition $(q, \mathsf{Z}) \xRightarrow{b} (q', \mathsf{Z}')$ for $\mathsf{Z}' = \text{local-elapse}([R](\mathsf{Z} \cap \mathsf{Z}_g \cap \mathsf{Z}_{sync}))$ where $\mathsf{Z}_g = \bigcap_{p \in dom(b)} \mathsf{Z}_{g_p}$, $\mathsf{Z}_{sync} = \{\mathsf{v} \mid \mathsf{v}(t_{p_1}) = \mathsf{v}(t_{p_2}) \text{ for } p_1, p_2 \in dom(b)\}$ and $R = [\bigcup_{p \in dom(b)} R_p]$. Intuitively, Z' is the set of valuations obtained through reset and then local-time elapse, from valuations in Z that satisfy the guard and such that the processes involved in action b are synchronised. We extend $\xrightarrow{b}$ to $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$ for a sequence of actions u in the obvious way.

Using local zones, we construct a local zone graph and show that it is sound and complete for reachability testing. The only missing step is to verify the pre/post properties of runs on local zones. We say a local zone is *time-elapsed* if $\mathsf{Z} = \text{local-elapse}(\mathsf{Z})$.

▶ **Lemma 12** (Pre and post properties of runs on local zones). *Let u be a sequence of actions.*

- *If $(q, \mathsf{v}) \xrightarrow{u} (q', \mathsf{v}')$ and $\mathsf{v} \in \mathsf{Z}$ for some time-elapsed local zone Z then $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$ and $\mathsf{v}' \in \mathsf{Z}'$ for some local zone Z'.*
- *If $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$ and $\mathsf{v}' \in \mathsf{Z}'$ then $(q, \mathsf{v}) \xrightarrow{u} (q', \mathsf{v}')$, for some $\mathsf{v} \in \mathsf{Z}$.*

▶ **Definition 13** (Local zone graph). *For a network of timed automata $\mathcal{N}$ the* local zone graph *of $\mathcal{N}$, denoted* $\text{LZG}(\mathcal{N})$*, is a transition system whose nodes are of the form (q, Z) where Z is a time elapsed local zone, and whose transitions are steps $(q, \mathsf{Z}) \xrightarrow{b} (q', \mathsf{Z}')$. The initial node $(q^{init}, \mathsf{Z}^{init})$ consists of the initial state q_{init} of the network and the local zone* $\mathsf{Z}^{init} = \text{local-elapse}(\{\mathsf{v}^{init}\})$*.*

Directly from Lemma 12 we obtain the main property of local zone graphs stated in [4]. This allows us to use local zone graphs for reachability testing.

▶ **Theorem 14.** *For a given network of timed automata $\mathcal{N}$, there is a run of the network reaching a state q iff for some non-empty local zone Z, node (q, Z) is reachable in* $\text{LZG}(\mathcal{N})$ *from its initial node.*

Notice that $\text{LZG}(\mathcal{N})$ may still be infinite and it cannot be used directly for reachability checking. The solution in Remark 11 does not apply to local zones due to the multiple reference clocks. This problem will be addressed in Section 5. We first focus on important properties of the local-time zone graph w.r.t. concurrency.

4 Why are local zone graphs better than global zone graphs?

The important feature about local zone graphs, as noticed in [4], is that two transitions on actions with disjoint domains commute (see Figure 1).

▶ **Lemma 15** (Commutativity on local zones [4]). *Suppose $dom(a) \cap dom(b) = \emptyset$. If $(q, \mathsf{Z}) \xrightarrow{ab} (q', \mathsf{Z}')$ then $(q, \mathsf{Z}) \xrightarrow{ba} (q', \mathsf{Z}')$.*

From the above lemma, we get the following property.

▶ **Corollary 16.** *If $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$ and $u \sim w$, then $(q, \mathsf{Z}) \xrightarrow{w} (q', \mathsf{Z}')$*

Thus starting from a local zone, all equivalent interleavings of a sequence of actions u end up in the same local zone. This is in stark contrast to the global zone graph, where each interleaving results in a possibly different global zone. Let

$$MZ(q, Z, u) = \{v' \mid \exists v \in Z,\ \exists w,\ w \sim u \text{ and } (q, v) \xRightarrow{w} (q', v')\}$$

denote the union of all these global zones. Salah et al. [17] have shown that, surprisingly, $MZ(q, Z, u)$ is always a global zone. We call it *aggregated zone*, and the notation MZ is in the memory of Oded Maler. In the same work, this observation was extended to an algorithm for *acyclic timed automata* that from time to time merged zones reached by equivalent paths to a single global zone. We prove below that this aggregated zone can, in fact, be obtained directly in the local zone graph: the aggregated (global) zone is exactly the set of synchronized valuations obtained after executing u in the local zone semantics. Here we need some notation: let Z be a global zone and Z a local zone; define $\mathrm{sync}(\mathsf{Z}) = \{\mathsf{v} \in \mathsf{Z} \mid \mathsf{v} \text{ is synchronized}\}$; $\mathsf{local}(Z) = \{\mathsf{local}(v) \mid v \in Z\}$ and $\mathsf{global}(\mathrm{sync}(\mathsf{Z})) = \{\mathsf{global}(\mathsf{v}) \mid \mathsf{v} \in \mathrm{sync}(\mathsf{Z})\}$.

▶ **Lemma 17.** *For every global zone Z and local zone Z:* $\mathrm{sync}(\mathsf{Z})$ *and* $\mathsf{local}(Z)$ *are local zones and* $\mathsf{global}(\mathrm{sync}(\mathsf{Z}))$ *is a global zone.*

Proof. $\mathrm{sync}(\mathsf{Z})$ is the local zone $\mathsf{Z} \wedge \bigwedge_{i,j}(t_i = t_j)$; $\mathsf{local}(Z)$ is the local zone obtained by replacing t with some t_i in each constraint, and adding the constraints $\bigwedge_{i,j}(t_i = t_j)$; $\mathsf{global}(\mathrm{sync}(\mathsf{Z}))$ is obtained by replacing each t_i with t in each constraint of Z. ◀

▶ **Theorem 18.** *Consider a state q, a sequence of actions u and a time elapsed global zone Z. Consider the local zone $\mathsf{Z} = \text{local-elapse}(\mathsf{local}(Z))$. If $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$, we have $MZ(q, Z, u) = \mathsf{global}(\mathrm{sync}(\mathsf{Z}'))$, otherwise $MZ(q, Z, u) = \emptyset$.*

Proof. Pick $v' \in MZ(q, Z, u)$. There exists $w \sim u$, $v \in Z$ and a global run $(q, v) \xRightarrow{w} (q', v')$. From Lemma 10, there exists a local run $(q, \mathsf{local}(v)) \xrightarrow{w} (q', \mathsf{local}(v'))$. By assumption, $\mathsf{local}(v) \in \mathsf{Z}$. Hence from the pre property of local zones (Lemma 12), there exists $(q, \mathsf{Z}) \xrightarrow{w} (q', \mathsf{Z}_w)$ such that $\mathsf{local}(v') \in \mathsf{Z}_w$. As $\mathsf{local}(v')$ is synchronized, we get $\mathsf{local}(v') \in \mathrm{sync}(\mathsf{Z}_w)$. But, by Corollary 16, $\mathsf{Z}_w = \mathsf{Z}'$. This proves $\mathsf{local}(v') \in \mathrm{sync}(\mathsf{Z}')$ and hence $v' \in \mathsf{global}(\mathrm{sync}(\mathsf{Z}'))$.

For the other direction take $v' \in \mathsf{global}(\mathrm{sync}(\mathsf{Z}'))$. As $(q, \mathsf{Z}) \xrightarrow{u} (q', \mathsf{Z}')$, by post property of local zones (Lemma 12) there is a local run $(q, \mathsf{v}_u) \xrightarrow{u} (q', \mathsf{local}(v'))$ for some $\mathsf{v}_u \in \mathsf{Z}$. Since $\mathsf{v}_u \in \mathsf{Z}$, it is obtained by a local time elapse from some $\mathsf{v} \in \mathsf{local}(Z)$. Hence v is synchronized and $\mathsf{global}(\mathsf{v}) \in Z$. From Lemma 10 we get that for some $w \sim u$ there is a global run $(q, \mathsf{global}(\mathsf{v})) \xRightarrow{w} (q', v')$. Hence $v' \in MZ(q, Z, u)$. ◀

Theorem 18 gives an efficient way to compute aggregated zones: it is sufficient to compute local zone graphs. Computing local zone graphs is not more difficult than computing global zone graphs. But, surprisingly, the combinatorial explosion due to interleaving does not occur in local zone graphs, thanks to the theorem above. Hence, this gives an incentive to work with local zone graphs instead of global zone graphs.

This contrasts with the aggregation algorithm in [17] which requires to store all the paths to a global zone and detect situations where zones can be merged, that is, when all the equivalent permutations have been visited. Another important limitation of the algorithm in [17] is that it can only be applied to acyclic zone graphs. If local zone graphs can be computed for general timed automata (which contain cycles), we can get to use the aggregation feature for all networks (and not only acyclic ones). To do this, there is still a major problem left: local zone graphs could be infinite when the automata contain cycles.

5 Making local zone graphs finite

The standard approach to make a global zone graph finite is to use a subsumption operation between global zones [2, 12]. Such a subsumption operation is usually based on a finite index simulation between (global) valuations parameterized by certain maximum constants occurring in guards. We first discuss technical problems that arise when we lift these simulations to local valuations. In the paper introducing local time semantics and local zone graphs [4] a notion of a *catch-up equivalence* between local valuations is defined. This equivalence is a finite index simulation. So one could, in theory, construct a finite local zone graph using catch-up subsumption as a finite abstraction. Unfortunately, the question of effective algorithms for catch-up subsumption was left open in [4], and, to the best of our knowledge, there is no efficient procedure for catch-up subsumption.

Another finite abstraction of the local zone graph was proposed by Minea [15, 16]. We however believe that Minea's approach carries a flaw, and a different idea is needed to get finiteness. Minea's approach is founded on an equivalence between local valuations in the lines of the region equivalence [1]. Let $c_{\max}$ be the maximum constant appearing among the guards of a timed automata network. Two local valuations v_1 and v_2 are said to be equivalent, written as $\mathsf{v}_1 \simeq_{\text{reg}} \mathsf{v}_2$ if for all variables $\alpha, \beta \in \widetilde{X} \cup \{t_1, \dots, t_k\}$ (note that all reference clocks are included):

- either $\lfloor \mathsf{v}_1(\alpha) - \mathsf{v}_1(\beta) \rfloor = \lfloor \mathsf{v}_2(\alpha) - \mathsf{v}_2(\beta) \rfloor$,
- or $\lfloor \mathsf{v}_1(\alpha) - \mathsf{v}_1(\beta) \rfloor > c_{\max}$ and $\lfloor \mathsf{v}_2(\alpha) - \mathsf{v}_2(\beta) \rfloor > c_{\max}$,
- or $\lfloor \mathsf{v}_1(\alpha) - \mathsf{v}_1(\beta) \rfloor < -c_{\max}$ and $\lfloor \mathsf{v}_2(\alpha) - \mathsf{v}_2(\beta) \rfloor < -c_{\max}$.

It is claimed (in Proposition 6 of [15]) that this equivalence is preserved over local time elapse: for every process p, and for $\delta \geq 0$ there exists $\delta' \geq 0$ such that $\mathsf{v}_1 +_p \delta \simeq_{\text{reg}} \mathsf{v}_2 +_p \delta'$. However, this is not true, as we now exhibit a counter-example. Consider 2 processes with clocks $X_1 = \{x\}$, $X_2 = \{y\}$. This gives $\widetilde{X} = \{\widetilde{x}, \widetilde{y}\}$ and $T = \{t_1, t_2\}$. Let $c_{\max} = 3$. Define local valuations $\mathsf{v}_1 : \widetilde{x} = 0, t_1 = 0, \widetilde{y} = 0, t_2 = 4$ and $\mathsf{v}_2 : \widetilde{x} = 0, t_1 = 0, \widetilde{y} = 0, t_2 = 5$.

Note that the differences in v_1 are either $0, 4$ or -4 and the corresponding differences in v_2 are $0, 5$ or -5. Hence by definition, $\mathsf{v}_1 \simeq_{\text{reg}} \mathsf{v}_2$. Consider valuation $\mathsf{v}_1 +_1 2$ obtained from v_1 by local delay of 2 units in component 1, that is $\mathsf{v}_1 +_1 2 : \widetilde{x} = 0, t_1 = 2, \widetilde{y} = 0, t_2 = 4$ Observe that in $\mathsf{v}_1 +_1 2$, the difference $t_1 - \widetilde{x} = 2$ and $t_2 - t_1 = 2$ which are both smaller than $c_{\max}$. We claim there is no local delay δ' such that $\mathsf{v}_1 +_1 2 \simeq_{\text{reg}} \mathsf{v}_2 +_1 \delta'$. Valuation $\mathsf{v}_2 +_1 \delta'$ is given by $\widetilde{x} = 0, t_1 = \delta', \widetilde{y} = 0, t_2 = 5$. If $\mathsf{v}_1 +_1 2 \simeq_{\text{reg}} \mathsf{v}_2 +_1 \delta'$, we need $\delta' = 2$ (due to difference $t_1 - \widetilde{x}$) and $5 - \delta' = 2$ (due to difference $t_2 - t_1$). This is not possible.

The main problem is that the above equivalence "forgets" actual values when the difference between reference clocks is above c_{max}. Even if this difference is bigger than the maximum constant, local delays can bring them within the constant c_{max}. Such a situation does not arise in the global semantics, as there is a single reference clock.

In [15] a widening operator on local zones based on $\simeq_{\text{reg}}$ is used for finiteness: given a canonical representation of a zone Z, the *maximized zone* with respect to $c_{\max}$ is obtained by changing every constraint $y_1 - y_2 \triangleleft c$ to $y_1 - y_2 < \infty$ if $c > c_{\max}$, and to $y_1 - y_2 < -c_{\max}$ if $c < -c_{\max}$. In the local zone graph construction, each local zone is maximized and inclusion between maximized zones is used for termination. Figure 2 gives an example of a network $\langle A_1, A_2 \rangle$ where the maximized local zone graph is unsound. This is shown by making use of the valuations v_1 and v_2 above. We also add an extra clock z in component A_2 for convenience. Note that $c_{\max} = 3$. Although clock y does not appear in A_2, one can assume that there are other transitions from q_0 that deal with y (we avoid illustrating these transitions explicitly). In the discussion below, $\mathsf{v}_1, \mathsf{v}_2$ are valuations restricted to $\widetilde{x}, \widetilde{y}, t_1$ and t_2. In order to reach the state p_2, the synchronization action c needs to be taken: transition sequence $a_1 c$ requires c to be taken at global time 4, and transition sequence $b_1 b_2 c$ requires c at global time 5. Hence c is not enabled in the network. This is witnessed by c not being enabled in the local zone graph (middle picture). Valuation v_2 is present in the zone reached after $b_1 b_2$.

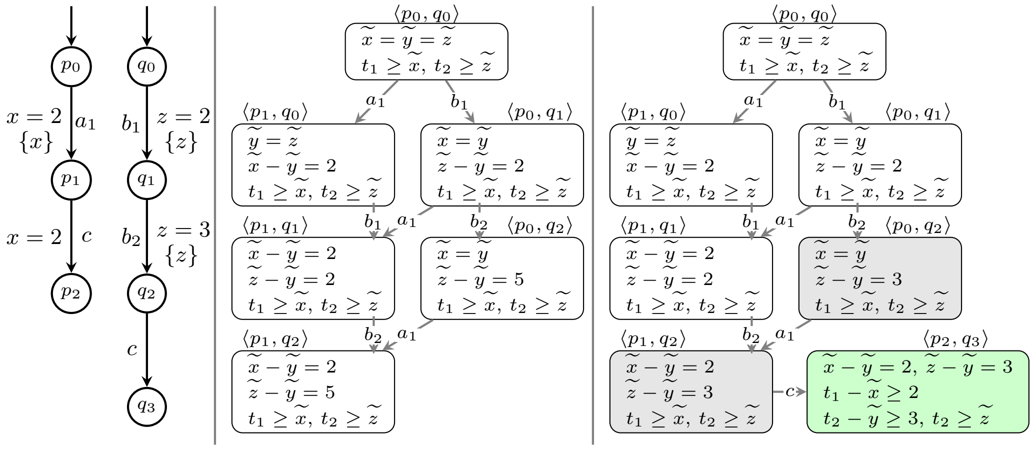

p_0
$x = 2$ a_1
$\{x\}$
p_1
$x = 2$ c
p_2
q_0
b_1 $z = 2$
$\{z\}$
q_1
b_2 $z = 3$
$\{z\}$
q_2
c
q_3
$\langle p_0, q_0 \rangle$ $\tilde{x} = \tilde{y} = \tilde{z}$, $t_1 \geq \tilde{x}$, $t_2 \geq \tilde{z}$
a_1
b_1
$\langle p_1, q_0 \rangle$ $\tilde{y} = \tilde{z}$, $\tilde{x} - \tilde{y} = 2$, $t_1 \geq \tilde{x}$, $t_2 \geq \tilde{z}$
$\langle p_0, q_1 \rangle$ $\tilde{x} = \tilde{y}$, $\tilde{z} - \tilde{y} = 2$, $t_1 \geq \tilde{x}$, $t_2 \geq \tilde{z}$
b_1 a_1 b_2
$\langle p_1, q_1 \rangle$ $\tilde{x} - \tilde{y} = 2$, $\tilde{z} - \tilde{y} = 2$, $t_1 \geq \tilde{x}$, $t_2 \geq \tilde{z}$
$\langle p_0, q_2 \rangle$ $\tilde{x} = \tilde{y}$, $\tilde{z} - \tilde{y} = 5$, $t_1 \geq \tilde{x}$, $t_2 \geq \tilde{z}$
b_2 a_1
$\langle p_1, q_2 \rangle$ $\tilde{x} - \tilde{y} = 2$, $\tilde{z} - \tilde{y} = 5$, $t_1 \geq \tilde{x}$, $t_2 \geq \tilde{z}$
$\langle p_0, q_0 \rangle$ $\tilde{x} = \tilde{y} = \tilde{z}$, $t_1 \geq \tilde{x}$, $t_2 \geq \tilde{z}$
a_1
b_1
$\langle p_1, q_0 \rangle$ $\tilde{y} = \tilde{z}$, $\tilde{x} - \tilde{y} = 2$, $t_1 \geq \tilde{x}$, $t_2 \geq \tilde{z}$
$\langle p_0, q_1 \rangle$ $\tilde{x} = \tilde{y}$, $\tilde{z} - \tilde{y} = 2$, $t_1 \geq \tilde{x}$, $t_2 \geq \tilde{z}$
b_1 a_1 b_2
$\langle p_1, q_1 \rangle$ $\tilde{x} - \tilde{y} = 2$, $\tilde{z} - \tilde{y} = 2$, $t_1 \geq \tilde{x}$, $t_2 \geq \tilde{z}$
$\langle p_0, q_2 \rangle$ $\tilde{x} = \tilde{y}$, $\tilde{z} - \tilde{y} = 3$, $t_1 \geq \tilde{x}$, $t_2 \geq \tilde{z}$
b_2 a_1
$\langle p_1, q_2 \rangle$ $\tilde{x} - \tilde{y} = 2$, $\tilde{z} - \tilde{y} = 3$, $t_1 \geq \tilde{x}$, $t_2 \geq \tilde{z}$
c
$\langle p_2, q_3 \rangle$ $\tilde{x} - \tilde{y} = 2$, $\tilde{z} - \tilde{y} = 3$, $t_1 - \tilde{x} \geq 2$, $t_2 - \tilde{y} \geq 3$, $t_2 \geq \tilde{z}$

Figure 2 *Left:* network $\langle A_1, A_2 \rangle$; *Middle:* local zone graph; *Right:* maximized local zone graph of [15]. State (p_2, q_3) is not reachable in local zone graph, but becomes reachable after maximization.

The maximized local zone graph is shown on the right. Zones where maximization makes a difference are shaded gray. In particular, the zone $b_1 b_2$ on maximization adds valuation v_1, from which $a_1 c$ is enabled, giving a zone in the maximized local zone graph with state p_2.

5.1 Synchronized valuations for subsumption

A finite abstraction of the differences between reference clocks constitutes the main challenge in obtaining a finite local zone graph. We propose a different solution which bypasses the need to worry about such differences: *restrict to synchronized valuations for subsumption.*

Subsumptions over global zones are well studied [1, 2, 12]. Taking an off-the-shelf finite abstraction $\mathfrak{a}$ and a subsumption $\sqsubseteq^{\mathfrak{a}}$ between global zones, we want to do the following: given two local zones Z_1 and Z_2 we perform a subsumption test $\mathsf{global}(\text{sync}(\mathsf{Z}_1)) \sqsubseteq^{\mathfrak{a}} \mathsf{global}(\text{sync}(\mathsf{Z}_2))$. By finiteness of the abstraction, we will get only finitely many local zones Z with incomparable $\mathsf{global}(\text{sync}(\mathsf{Z}))$. In order to do this, we need the crucial fact that $\mathsf{global}(\text{sync}(\mathsf{Z}))$ is a zone (shown in Lemma 17). Given two configurations $s := (q, \mathsf{Z})$ and $s' := (q', \mathsf{Z}')$ of the local zone graph, we write $s \sqsubseteq^{\mathfrak{a}}_{sync} s'$ if $q = q'$ and $\mathsf{global}(\text{sync}(\mathsf{Z})) \sqsubseteq^{\mathfrak{a}} \mathsf{global}(\text{sync}(\mathsf{Z}'))$.

▶ **Definition 19** (Local sync graph). *Let $\mathfrak{a}$ be a finite abstraction over global zones. A* local sync graph *G of a network of timed automata $\mathcal{N}$ based on $\mathfrak{a}$, is a tree and a subgraph of* LZG($\mathcal{N}$) *satisfying the following conditions:*

C0 *every node of G is labeled either* covered *or* uncovered*;*
C1 *the initial node of* LZG($\mathcal{N}$) *belongs to G and is labeled uncovered;*
C2 *every node is reachable from the initial node;*
C3 *for every uncovered node s, all its successor transitions $s \xrightarrow{a} s'$ occurring in* LZG($\mathcal{N}$) *should be present in G;*
C4 *for every covered node $s \in G$ there is an uncovered node $s' \in G$ such that $s \sqsubseteq^{\mathfrak{a}}_{sync} s'$. A covered node has no successors.*

The above definition essentially translates to this algorithm: explore the local zone graph say in a BFS fashion, and subsume (cover) using $\sqsubseteq^{\mathfrak{a}}_{sync}$ (similar to algorithm for global zone graph as described in page 8). Local sync graphs are not unique, since the final graph depends on the order of exploration. Every local sync graph based on a finite abstraction $\mathfrak{a}$ is finite. Theorem 20 below states that local sync graphs are sound and complete for reachability, and this algorithm is correct.

▶ **Theorem 20** (Soundness and completeness of local sync graphs). *A state q is reachable in a network of timed automata $\mathcal{N}$ iff a node (q, Z), with Z non-empty, is reachable from the initial node in a local sync graph for $\mathcal{N}$.*

Proof. If (q, Z) is reachable in a local sync graph then it is trivially reachable in the local zone graph and the (backward) implication follows from soundness of local zone graphs (Theorem 14). For the other direction which proves completeness of local sync graphs, let us take a global run: $(q_{init}, v_{init}) \xRightarrow{\delta_1, b_1} (q_1, v_1) \cdots \xRightarrow{\delta_n, b_n} (q_n, v_n)$. Take a local sync graph G, based on abstraction $\mathfrak{a}$ coming from a simulation $\preccurlyeq$. By induction on i, for every (q_i, v_i) we will find an uncovered node (q_i, Z_i) of G, and a synchronized local valuation $\mathsf{v}_i \in \mathsf{Z}_i$ such that $v_i \preccurlyeq \mathsf{global}(\mathsf{v}_i)$. This proves completeness, since every reachable (q_i, v_i) will have a representative node in the local sync graph.

The induction base is immediate, so let us look at the induction step. Consider the global step $(q_i, v_i) \xRightarrow{\delta_i, b_i} (q_{i+1}, v_{i+1})$. Since $v_i \preccurlyeq \mathsf{global}(\mathsf{v}_i)$, there is a delay δ'_i such that $(q_i, \mathsf{global}(\mathsf{v}_i)) \xRightarrow{\delta'_i, b_i} (q_{i+1}, v'_{i+1})$ and $v_{i+1} \preccurlyeq v'_{i+1}$. As the global delay δ'_i can be thought of a sequence of local delays, we have local run $(q_i, \mathsf{v}_i) \xrightarrow{b_i} (q_{i+1}, \mathsf{v}'_{i+1})$, where $\mathsf{v}'_{i+1} = \mathsf{local}(v'_{i+1})$. Note that v'_{i+1} is synchronized and $v_{i+1} \preccurlyeq \mathsf{global}(\mathsf{v}'_{i+1})$. From the pre-property of local zones (Lemma 12) there exists a transition $(q_i, \mathsf{Z}_i) \xrightarrow{b_i} (q_{i+1}, \mathsf{Z}'_{i+1})$ with $\mathsf{v}'_{i+1} \in \mathsf{Z}'_{i+1}$, in fact, $\mathsf{v}'_{i+1} \in \mathsf{sync}(\mathsf{Z}'_{i+1})$. If $(q_{i+1}, \mathsf{Z}'_{i+1})$ is uncovered, take v'_{i+1} for v_{i+1} and Z'_{i+1} for Z_{i+1} (needed by the induction step). Otherwise, from condition C4, there is an uncovered node $(q_{i+1}, \mathsf{Z}''_{i+1})$ such that $\mathsf{global}(\mathsf{sync}(\mathsf{Z}'_{i+1})) \preccurlyeq \mathsf{global}(\mathsf{sync}(\mathsf{Z}''_{i+1}))$. This gives $\mathsf{v}''_{i+1} \in \mathsf{sync}(\mathsf{Z}''_{i+1})$ such that $\mathsf{global}(\mathsf{v}'_{i+1}) \preccurlyeq \mathsf{global}(\mathsf{v}''_{i+1})$. Now take v''_{i+1} for v_{i+1} and Z''_{i+1} for Z_{i+1}. ◀

6 Experiments

We have implemented the construction of local sync graphs in our prototype TChecker [11] and compared it with two implementations of the usual global zone graph method: TChecker and UPPAAL [13, 3], the state-of-the-art verification tool for timed automata. The three implementations use a breadth-first search with subsumption, and the $\sqsubseteq^{\mathfrak{a}}_{sync}$ subsumption in the case of local sync graph. Table 1 presents results of our experiments on standard models from the literature (except "Parallel" that is a model we have introduced).

Local sync graphs yield no gain on 3 standard examples (which are not given in Table 1): "CSMA/CD", "FDDI" and "Fischer". In these models, the three algorithms visit and store the same number of nodes. The reason is that for "CSMA/CD" and "FDDI", replacing each local zone Z in the local zone graph by its set of synchronized valuations $sync(\mathsf{Z})$ yields exactly the zone graph. In the third model, "Fischer", every control state appears at most once in the global zone graph. So there is no hope to achieve any gain with our technique. This is due to the fact that doing ab or ba results in two different control states in the automaton. So "Fischer" is out of the scope of our technique.

In contrast, we observe significant improvements on other standard models (Table 1). Observe that due to subsumption, the order in which nodes are visited impacts the total number of visited nodes. UPPAAL and our prototype TChecker (Global ZG column) may not visit the same number of nodes despite the fact that they implement the same algorithm. In our prototype we use the same order of exploration for Global ZG and Local ZG. "CorSSO" and "Critical region" are standard examples from the literature. "Dining Philosophers" is a modification of the classical problem where a philosopher releases her left fork if she cannot take her right fork within a fixed amount of time [14]. "Parallel" is a model we have introduced, where concurrent processes compete to access a resource in mutual exclusion.

Table 1 Experimental results obtained by running UPPAAL and our prototype TChecker (Global ZG and Local ZG) on a MacBook Pro 2013 with 4 2.4GHz Intel Core i5 and 16 GB of memory. The timeout is 90 seconds. For each model we report the number of concurrent processes.

Models	UPPAAL			Global ZG			Local ZG		
(# processes)	visited	stored	sec.	visited	stored	sec.	visited	stored	sec.
CorSSO 3	64378	61948	1.48	64378	61948	1.41	1962	1962	0.05
CorSSO 4	timeout			timeout			23784	23784	0.69
CorSSO 5	timeout			timeout			281982	281982	16.71
Critical reg. 4	78049	53697	1.45	75804	53697	2.27	44490	28400	2.40
Critical reg. 5	timeout			timeout			709908	389614	75.55
Dining Phi. 7	38179	38179	34.61	38179	38179	7.28	2627	2627	0.32
Dining Phi. 8	timeout			timeout			8090	8090	1.65
Dining Phi. 9	timeout			timeout			24914	24914	7.10
Dining Phi. 10	timeout			timeout			76725	76725	30.20
Parallel 6	11743	11743	4.82	11743	11743	1.09	256	256	0.02
Parallel 7	timeout			timeout			576	576	0.04
Parallel 8	timeout			timeout			1280	1280	0.11

We observe an order of magnitude gains for most of these four models. The reason is that in most states when two processes can perform actions a and b, doing ab or ba leads to the same control-state of the automaton. Hence, a difference between ab and ba (if any) is encoded in distinct zones Z_{ab} and Z_{ba} obtained along these two paths in the global zone graph. In contrast, the two paths result in the same zone Z (containing both Z_{ab} and Z_{ba}) in the local sync graph. In consequence, our approach that combines the local zone graph and abstraction using synchronized zones is very efficient in this situation.

7 Conclusions

We have revisited local-time semantics of timed automata and local zone graphs. We have discovered a very useful fact that local zones calculate aggregated zones: global zones that are unions of all the zones obtained by equivalent executions [17]. We have used this fact as a theoretical foundation for an algorithm constructing local zone graphs and using subsumption on aggregated zones at the same time.

We have shown that, unfortunately, subsumption operations on local zones proposed in the literature do not work. We have proposed a new subsumption for local zone graphs based on standard abstractions for timed automata, applied on synchronized zones. The restriction to synchronized zones is crucial as standard abstractions cannot handle multiple reference clocks. A direction for future work is to find abstractions for local zones.

Our algorithm is the first efficient implementation of local time zone graphs and aggregated zones. Experimental results show an order of magnitude gain with respect to state-of-the-art algorithms on several standard examples.

As future work, we plan to develop partial-order techniques taking advantage of the high level of commutativity in local zone graphs. Existing methods are not directly applicable in the timed setting. In particular, contrary to expectations, actions with disjoint domains may not be independent (in a partial order sense) in a local zone graph [15]. Thus, it will be very interesting to understand the structure of local zone graphs better. A recent partial-order method proposed for timed-arc Petri nets [6] gives a hope that such obstacles can be overcome. For timed networks with cycles, the interplay of partial-order and subsumption adds another level of difficulty.

References

1 Rajeev Alur and David L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126:183–235, 1994.

2 Gerd Behrmann, Patricia Bouyer, Kim G. Larsen, and Radek Pelánek. Lower and Upper Bounds in Zone-Based Abstractions of Timed Automata. *International Journal on Software Tools for Technology Transfer*, 8(3):204–215, 2006.

3 Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Hakansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. UPPAAL 4.0. In *QEST*, pages 125–126. IEEE Computer Society, 2006.

4 Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial Order Reductions for Timed Systems. In *CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 485–500, 1998.

5 Johan Bengtsson and Wang Yi. Timed Automata: Semantics, Algorithms and Tools. In *ACPN 2003*, volume 3098 of *Lecture Notes in Computer Science*, pages 87–124. Springer, 2003.

6 Frederik M. Bønneland, Peter Gjøl Jensen, Kim Guldstrand Larsen, Marco Muñiz, and Jirí Srba. Start Pruning When Time Gets Urgent: Partial Order Reduction for Timed Systems. In *CAV*, volume 10981 of *Lecture Notes in Computer Science*, pages 527–546. Springer, 2018.

7 Conrado Daws and Stavros Tripakis. Model Checking of Real-Time Reachability Properties Using Abstractions. In *TACAS*, volume 1384 of *Lecture Notes in Computer Science*, pages 313–329. Springer, 1998.

8 D. L. Dill. Timing Assumptions and Verification of Finite-state Concurrent Systems. In *Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems*, pages 197–212, 1990.

9 Paul Gastin, Sayan Mukherjee, and B. Srivathsan. Fast algorithms for handling diagonal constraints in timed automata. *CoRR*, abs/1904.08590, 2019. `arXiv:1904.08590`.

10 R Govind, Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Revisiting local time semantics for networks of timed automata. *CoRR*, abs/1907.02296, 2019. `arXiv:1907.02296`.

11 F. Herbreteau and G. Point. TChecker. `https://github.com/fredher/tchecker`, v0.2 - April 2019.

12 Frédéric Herbreteau, B. Srivathsan, and Igor Walukiewicz. Better Abstractions for Timed Automata. In *LICS*, pages 375–384. IEEE Computer Society, 2012.

13 Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL in a nutshell. *STTT*, 1(1-2):134–152, 1997.

14 Denis Lugiez, Peter Niebert, and Sarah Zennou. A partial order semantics approach to the clock explosion problem of timed automata. *Theor. Comput. Sci.*, 345(1):27–59, 2005.

15 Marius Minea. Partial Order Reduction for Model Checking of Timed Automata. In *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*, pages 431–446. Springer, 1999.

16 Marius Minea. *Partial Order Reduction for Verification of Timed Systems*. PhD thesis, School of Computer Science, Carnegie Mellon University Pittsburgh, PA 15213, 1999.

17 Ramzi Ben Salah, Marius Bozga, and Oded Maler. On Interleaving in Timed Automata. In *CONCUR*, volume 4137 of *Lecture Notes in Computer Science*, pages 465–476. Springer, 2006.

18 Marcelo Sousa, César Rodríguez, Vijay D'Silva, and Daniel Kroening. Abstract Interpretation with Unfoldings. In *CAV*, pages 197–216, 2017.

Approximate Learning of Limit-Average Automata

Jakub Michaliszyn
University of Wrocław, Poland
jmi@cs.uni.wroc.pl

Jan Otop
University of Wrocław, Poland
jotop@cs.uni.wroc.pl

Abstract

Limit-average automata are weighted automata on infinite words that use average to aggregate the weights seen in infinite runs. We study approximate learning problems for limit-average automata in two settings: passive and active. In the passive learning case, we show that limit-average automata are not PAC-learnable as samples must be of exponential-size to provide (with good probability) enough details to learn an automaton. We also show that the problem of finding an automaton that fits a given sample is NP-complete. In the active learning case, we show that limit-average automata can be learned almost-exactly, i.e., we can learn in polynomial time an automaton that is consistent with the target automaton on almost all words. On the other hand, we show that the problem of learning an automaton that approximates the target automaton (with perhaps fewer states) is NP-complete. The abovementioned results are shown for the uniform distribution on words. We briefly discuss learning over different distributions.

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

Keywords and phrases weighted automata, learning, expected value

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.17

Related Version A full version of the paper is available at `https://arxiv.org/abs/1906.11104`.

Funding The National Science Centre (NCN), Poland under grant 2017/27/B/ST6/00299.

1 Introduction

Quantitative verification has been proposed to verify non-Boolean system properties such as performance, energy consumption, fault-tolerance, etc. There are two main challenges in applying quantitative verification in practice. First, formalization of quantitative properties is difficult as specifications are given directly as weighted automata, which extend finite automata with weights on transitions [18, 15]. Quantitative logics, which ease specification, have either limited expression power or their model-checking problem is undecidable [11, 13]. Second, there is little research on abstraction in the quantitative setting [14], which would allow us to reduce the size of quantitative models, presented as weighted automata.

We approach both problems using learning of weighted automata. We can apply the learning framework to facilitate writing quantitative specifications and make it more accessible to non-experts. For abstraction purposes, we study approximate learning, having in mind that approximation of a system can significantly reduce its size.

We focus on weighted automata over infinite words which compute the long-run average of weights [15]. Such automata express interesting quantitative properties [15, 23] and admit polynomial-time algorithms for basic decision questions [15]. Some of the interesting properties can be only expressed by non-deterministic automata [15], but every non-deterministic weighted automaton is approximated by some deterministic weighted automaton on almost all words [29]. This means that, by allowing some small margin of error, we can focus on

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 17; pp. 17:1–17:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

deterministic automata while being able to model important system properties, such as minimal response time, minimal number of errors, the edit distance problem [23], and the *specification repair* framework from [10].

We therefore focus on approximate learning under probabilistic semantics, which corresponds to the average-case analysis in quantitative verification [16]. We treat words as random events and functions defined by weighted automata as random variables. In this setting, an automaton $\mathcal{A}'$ ϵ-approximates $\mathcal{A}$ if the expected difference between $\mathcal{A}$ and $\mathcal{A}'$ (over all words) is bounded by ϵ. We consider two learning approaches: passive and active.

In passive learning, we think of an automaton as a black-box. This might be a program, a specification, a working correct system to be replaced or abstracted, or even only some examples of how the automaton should work. Our goal is to construct a weighted automaton based on this black-box model by observing only inputs and outputs of the model.

In active learning, we assume presence of an interactive *teacher* that reveals the values of given examples and verifies whether a provided automaton is as intended; if not, the teacher responses with a witness, which is a word showing the difference between the constructed automaton and the intended automaton. This does not necessary mean that the teacher is familiar with the weighted automata formalism – to provide a witness, they may simply run the constructed automaton in parallel with the black-box and, if at any point their behaviors differ, provide the appropriate input to the learning algorithm.

Our contributions. We start with a discussion on the setting for learning problems. The first step is to find a suitable representation for samples; not all infinite words have finite representations. A natural idea is to use ultimately periodic words in samples; we study such samples. However, the probability distribution over ultimately periodic words is very different from the uniform distribution over infinite words. Therefore, we also consider samples consisting of a finite word u labeled with the expected value over all extensions of u. The probability distribution over such samples is closer to the uniform distribution over infinite words.

Then we study the passive learning problem. We show that for unique characterization of an automaton, we need a sample of exponential size. We study the complexity of the problem of finding an automaton that fits the whole sample. The problem, without additional restrictions, has a trivial and overfitting solution. To mitigate this, we impose bounds on automata size, and show that then the problem is NP-complete.

For active learning, we show that the problem of learning an almost-exact automaton can be solved in polynomial time, and finding an automaton of bounded size that approximates the target one cannot be done in polynomial time if $\mathrm{P} \neq \mathrm{NP}$. We conclude with a discussion on different probability distributions.

Related work. The *probably approximately correct* (PAC) learning, introduced in [35], is a general passive learning framework applied to various objects (DNF/CNF formulas, decision trees, automata, etc.) [26]. PAC learning of deterministic finite automata (DFA) has been extensively studied despite negative indicators. First, the *sample fitting* problem for DFA, where the task is to construct a minimal-size DFA consistent with a given sample, has been shown NP-complete [21]. Even approximate sample fitting, where we ask for a DFA at most polynomially greater than a miniaml-size DFA, remains NP-complete [34]. Second, it has been shown that existence of a polynomial-time PAC learning algorithm for DFA would break certain cryptographic systems (such as RSA) and hence it is unlikely [25]. Despite these negative results, it has been empirically shown that DFA can be efficiently learned [27]. In particular, if we assume *structural completeness* of a sample, then it determines a minimal DFA [32]. Pitt posed a question whether DFA are PAC-learnable under the uniform distribution [33], which remains open [2].

Angluin showed that DFA can be learned in polynomial time, if the learning algorithm can ask membership and equivalence queries [1]. This approach proved to be very fruitful and versatile. Angluin's algorithm has been adapted to learn NFA [12], automata over infinite words [3, 20], nominal automata over infinite alphabets [31], weighted automata over words [9] and weighted automata over trees [22, 28].

Recently, there has been a renewed interest in learning weighted automata [8, 28, 6, 5, 7]. These results apply to weighted automata over fields [18], which work over finite words. We, however, consider limit-average automata, which work over infinite words and cannot be defined using a field or even a semiring. Furthermore, we consider weighted (limit-average) automata under probabilistic semantics [16, 29], i.e., we consider functions represented by automata as random variables.

2 The setting

Given a finite alphabet Σ of letters, a *word* w is a finite or infinite sequence of letters. We denote the set of all finite words over Σ by Σ^*, and the set of all infinite words over Σ by Σ^ω. For a word w, we define $w[i]$ as the i-th letter of w, and we define $w[i,j]$ as the subword $w[i]w[i+1]\dots w[j]$ of w. We use the same notation for vectors and sequences; we assume that sequences start with 0 index.

A *deterministic finite automaton* (DFA) is a tuple $(\Sigma, Q, q_0, F, \delta)$ consisting of the alphabet Σ, a finite set of states Q, the initial state $q_0 \in Q$, a set of final states F, and a transition function $\delta \colon Q \times \Sigma \to Q$.

A (deterministic) $\textsc{LimAvg}$-automaton extends a DFA with a function $C \colon \delta \to \mathbb{Q}$ that defines rational *weights* of transitions. The size of such an automaton $\mathcal{A}$, denoted by $|\mathcal{A}|$, is the sum of the number of states of $\mathcal{A}$ and the lengths of binary encodings of all the weights.

A *run* π of a $\textsc{LimAvg}$-automaton $\mathcal{A}$ on a word w is a sequence of states $\pi[0]\pi[1]\dots$ such that $\pi[0]$ is the initial state and for every $i > 0$ we have $\delta(\pi[i-1], w[i]) = \pi[i]$. We do not consider ω-accepting conditions and assume that all infinite runs are accepting. Every run π of $\mathcal{A}$ on an infinite word w defines a sequence of weights $C(\pi)$ of successive transitions of $\mathcal{A}$, i.e., $C(\pi)[i] = C(\pi[i-1], w[i], \pi[i])$. The *value* of the run π is then defined as $\textsc{LimAvg}(\pi) = \limsup_{k\to\infty} \textsc{Avg}(C(\pi)[0,k])$, where for finite runs π we have $\textsc{Avg}(C(\pi)) = \textsc{Sum}(C(\pi))/|C(\pi)|$. The value of a word w assigned by the automaton $\mathcal{A}$, denoted by $\mathcal{L}_{\mathcal{A}}(w)$, is the value of the run of $\mathcal{A}$ on w.

We consider three classes of probability measures on words over the alphabet Σ.

- $\mathcal{U}^n$, for $n \in \mathbb{N}$, is the uniform probability distribution on Σ^n assigning each word the probability $|\Sigma|^{-n}$.
- $\mathcal{G}(\lambda)$, for a termination probability $\lambda \in \mathbb{Q}^+ \cap (0,1)$, is such that for $u \in \Sigma^*$ we have $\mathcal{G}(\lambda)(u) = |\Sigma|^{-|u|} \cdot (1-\lambda)^{|u|} \cdot \lambda$. Observe that the probability of words of the same length is equal, and the probability of generating a word of length k is $(1-\lambda)^k \cdot \lambda$. We can consider this as process generating finite words, which stops after each step with probability λ.
- $\mathcal{U}^\infty$ is the uniform probability measure on Σ^ω. Formally, we define $\mathcal{U}^\infty$ on basic sets $u\Sigma^\omega = \{uw \mid w \in \Sigma^\omega\}$ as follows: $\mathcal{U}^\infty(u\Sigma^\omega) = |\Sigma|^{-|u|}$. Then, $\mathcal{U}^\infty$ is the unique extension on all Borel sets in Σ^ω considered with the product topology [19, 4].

Automata as random variables. A weighted automaton $\mathcal{A}$ defines the measurable function $\mathcal{L}_{\mathcal{A}}(w) \colon \Sigma^\omega \mapsto \mathbb{R}$ that assigns values to words. We interpret such functions as random variables w.r.t. the probabilistic measure $\mathcal{U}^\infty$. Hence, for a given automaton $\mathcal{A}$, we consider the following quantities:

1. $\mathbb{E}(\mathcal{A})$ – the expected value of the random variable $\mathcal{L}_{\mathcal{A}}$ defined by $\mathcal{A}$ w.r.t. the uniform distribution $\mathcal{U}^\infty$ on Σ^ω, and
2. $\mathbb{E}(\mathcal{A} \mid u\Sigma^\omega)$ – the conditional expected value, defined for $\mathcal{U}^\infty$ as the expected value of $\mathcal{L}^u$ such that $\mathcal{L}^u(w) = \mathcal{L}_{\mathcal{A}}(uw)$.

We consider automata as generators of random variables; automata defining the same random variable are equivalent. Formally, LimAvg-automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are *almost equivalent* if and only if for almost all words w we have $\mathcal{L}_{\mathcal{A}_1}(w) = \mathcal{L}_{\mathcal{A}_2}(w)$. Note that almost all words means all except for words from some Y of probability 0.

LimAvg-automata considered over probability distributions are equivalent to Markov chains with the long-run average objectives presented in [4, Section 10.5.2].

▶ **Theorem 1** ([4]). *Let $\mathcal{A}$ be a LimAvg-automaton. (1) If $\mathcal{A}$ is strongly connected, then almost all words have the same value equal to $\mathbb{E}(\mathcal{A})$. (2) For almost all words w, the run on w eventually reaches some bottom strongly connected component (SCC) of $\mathcal{A}$.*

Theorem 1 has important consequences. First, we can contract each bottom SCC to a single state with self-loops of the same weight x being the expected value of that SCC. We refer to x as the value of that SCC. Such an operation does not affect almost equivalence. Second, while reasoning about LimAvg-automata, we can neglect all weights except for the values of bottom SCCs. We will omit weights other than the values of bottom SCCs.

Samples. A sample is a set of *labeled examples* of some (hidden) function $\mathcal{L} \colon \Sigma^\omega \to \mathbb{R}$. In the classical automata-learning approach words are finite, and hence they can be presented as examples along with the information whether the word belongs to the hidden language. Infinite-words, however, cannot be given directly to a learning algorithm. To mitigate this problem we consider two approaches: one is to restrict examples to ultimately periodic words, which have finite presentation, and the other is to consider finite words and ask for conditional expected values. We discuss both approaches below.

To distinguish samples with different types of labeled examples, we call them U-samples, E-samples and (E,n)-samples.

1. *Ultimately periodic words.* Consider an example being an ultimately periodic word uv^ω. It can presented as a pair of finite words (u,v) and we consider labeled examples (u,v,x), where $u, v \in \Sigma^*$ and $x = \mathcal{L}(uv^\omega)$. A set of labeled examples (u,v,x) is called an U-sample. To draw a random U-sample, we consider finite words u, v to be selected independently at random according to distributions $\mathcal{G}(\lambda_1)$ and $\mathcal{G}(\lambda_2)$ for some λ_1, λ_2. For such a set of pairs of words, we label them according to the function $\mathcal{L}$.
2. *Conditional expected values.* We consider examples, which are finite words $u \in \Sigma^*$. A labeled example is a pair (u,x), where $x = \mathbb{E}(\mathcal{L} \mid u\Sigma^\omega)$ is the conditional expected value of $\mathcal{L}$ under the condition that random words start with u. For such labeled examples we consider E-samples consisting of labeled words of various length, and (E,n)-samples consisting of words of length n. We assume that finite words for random E-samples are drawn according to a distribution $\mathcal{G}(\lambda)$ for some λ, and finite words for random (E,n)-samples are drawn according to the uniform distribution $\mathcal{U}^n$.

We only consider minimal consistent samples, i.e., samples that do not contain examples whose value can be computed from other examples in the sample. For instance, $\{(a,a,0),(a,aa,1)\}$ is an inconsistent U-sample, and $\{(aa,0),(ab,\frac{1}{2}),(a,1)\}$ is an inconsistent E-sample over $\{a,b\}$.

▶ Remark 2 (Incompatible distributions). Note that the distribution on ultimately periodic words differ from the uniform distribution on infinite words. The set of ultimately periodic words is countable and hence it has probability 0 (according to the distribution $\mathcal{U}^\infty$). Moreover, almost all infinite words contain all finite words as infixes, whereas this is not the case for ultimately periodic words under any probability distribution.

▶ Remark 3 (Feasibility of conditional expectation). Consider a LimAvg-automaton $\mathcal{A}$ computing $\mathcal{L}\colon \Sigma^\omega \to \mathbb{R}$. For a finite prefix u, we can compute $\mathbb{E}(\mathcal{L} \mid u\Sigma^\omega)$ in polynomial time in $|\mathcal{A}|$ [4]. If we consider $\mathcal{A}$ to be a black-box, which can be controlled, then $\mathbb{E}(\mathcal{L} \mid u\Sigma^\omega)$ can be approximated in the following way. We pick random words $v_1,\ldots,v_k$ of length k, compute partial averages in $\mathcal{A}$ of $uv_1,\ldots,uv_k$ and then take the average of these values. The probability that this process returns a value ϵ-close to $\mathbb{E}(\mathcal{L} \mid u\Sigma^\omega)$ converges to 1 at exponential rate with k.

3 Passive learning

Passive learning corresponds to a scenario with an uncontrolled working black-box system. The learner can only observe system's output, and its goal is to create an approximate model of the system. This task comprise of two problems. The first problem, *characterization*, is to assess whether the observations cover most, if not all, behaviors of the system. The second one, called *sample fitting*, is to create a reasonable automaton consistent with the observations. In this section we discuss both problems.

3.1 Characterization

A sample can cover only small part of the system. It is sometimes argued [27, 32], however, that if a sample is large enough, then it is likely to cover most, if not all, important behaviors. We show that for some LimAvg-automata, randomly drawn samples of size less than exponential are unlikely to demonstrate any probable values.

Let $||S||$ denote the sum of the lengths of all the examples in a sample S. A sample *distinguishes* two automata if it is consistent with exactly one of them. We show the following.

▶ **Theorem 4.** *For any n there are two automata $\mathcal{A}_n$, $\overline{\mathcal{A}}_n$ of size $n+4$ such that for almost all words w we have $|\mathcal{L}_{\mathcal{A}_n}(w)-\mathcal{L}_{\overline{\mathcal{A}}_n}(w)| = 2$, but a random U-sample, E-sample or (E,k)-sample (for any k) S distinguishes $\mathcal{A}_n$ and $\overline{\mathcal{A}}_n$ with the probability at most $\frac{||S||}{2^n}$.*

Proof. Consider the alphabet $\{a,b\}$ and $n \in \mathbb{N}$. We construct a LimAvg-automaton $\mathcal{A}_n$ with $n+4$ states. We use $n+2$ states to find the first occurrence of the infix $a^n b$ in the standard manner. When such an infix is found, the automaton moves to a state q_a if the following letter is a and to a state q_b otherwise. In q_a it loops with the weight 1 and in q_b it loops with the weight -1. All other weights are 0. So $\mathcal{A}_n$ returns 1 if the first occurrence of $a^n b$ if followed by a, -1 if it is followed by b, and 0 if there is no $a^n b$. The automaton $\overline{\mathcal{A}}_n$ has the same structure as $\mathcal{A}_n$, but the weights -1 and 1 are swapped.

We first observe that $\mathcal{A}_n$ and $\overline{\mathcal{A}}_n$ differ over almost all infinite words. Indeed, an infinite word with probability 1 contains the infix $a^n b$, and so on almost all words one of the automata returns -1 and the other one 1. Consider a sample S (it can be an E-sample, (E,k)-sample or U-sample). This sample distinguishes automata $\mathcal{A}_n$ and $\overline{\mathcal{A}}_n$ only if it contains an example with the infix $a^n b$ (in case of U-samples, this means that this infix occurs in uv of some example (u, v)) - all other examples for both automata are the same. The probability that S contains $a^n b$ as an infix of one of its examples is bounded by $\frac{||S||}{2^{n+1}}$. Indeed, the number of positions in all words is $||S||$ and the probability that $a^n b$ occurs on some specific position is at most $\frac{1}{2^{n+1}}$ for all types of samples and any $k > 0$ (if $k < n + 1$, the probability is 0). ◀

Therefore to be able to distinguish just two automata with probability $1 - \epsilon$, we need a sample such that $\frac{||S||}{2^{n+1}} > 1 - \epsilon$, which for fixed $\epsilon < 1$ is of exponential size. We show that the exponential upper bound is sufficient.

***U*-samples.** If automata $\mathcal{A}_1$, $\mathcal{A}_2$ of size n recognize different languages, then there is a word uw^ω such that $\mathcal{L}_{\mathcal{A}_1}(uw^\omega) \neq \mathcal{L}_{\mathcal{A}_2}(uw^\omega)$ and the length of u and w is bounded by n^2. Assume for simplicity that $\lambda = \lambda_1 = \lambda_2$. If the sample size is at least $|\Sigma|^{2n^2} \cdot \ln \frac{|\Sigma|^{2n^2}}{\epsilon} \cdot (1-\lambda)^{-2n^2} \cdot \lambda^{-2}$, then with probability $1 - \epsilon$ a random sample contains all such words, and so distinguishes all the automata of size n.

***E*-samples.** E-samples do not distinguish almost-equivalent automata, hence we cannot learn automata exactly. However, exponential samples are enough to learn automata up to almost equivalence. To see that consider two automata $\mathcal{A}_1$ and $\mathcal{A}_2$ of size n that are not almost equivalent. Due to Theorem 1 there is a word $u \in \Sigma^*$ such that u reaches bottom SCCs in both $\mathcal{A}_1$ and $\mathcal{A}_2$, and these bottom SCCs have different expected values. Using standard pumping argument, we can reduce the size of u to at most n^2. So if the sample size is at least $|\Sigma|^{n^2} \cdot \ln \frac{|\Sigma|^{n^2}}{\epsilon} \cdot (1 - \lambda)^{-n^2} \cdot \lambda^{-1}$, then with probability $1 - \epsilon$ it contains all words u of size n^2, and therefore distinguishes all automata of size n.

For (E, n)-samples, the reasoning is the similar, assuming n is quadratic in the size of the automaton: the sufficient sample size is $|\Sigma|^n \cdot \ln \frac{|\Sigma|^n}{\epsilon}$.

3.2 Consequences for PAC learning

We discuss the consequences of our results to the *probably approximately correct* (PAC) model of learning [35]. In the PAC framework, the learning algorithm should work independently of the probability distribution on samples. However, variants of the PAC framework have been considered where the distribution on samples is uniform [24]. In particular, PAC learning of DFA under the uniform distribution over words is a long-standing open problem [33, 2].

We restrict the classical PAC model and assume that observations are drawn according to the distributions $\mathcal{U}^n, \mathcal{G}(\lambda)$ (as discussed in Section 2) and the quality of the learned automaton is assessed using the uniform distribution over infinite words $\mathcal{U}^\infty$.

▶ **Problem 5** (PAC learning under fixed distributions). *Given $\epsilon, \delta \in \mathbb{Q}^+$, $n \in \mathbb{N}$ and an oracle returning random labeled examples consistent with some automaton $\mathcal{A}^T$ of size n, construct an automaton $\mathcal{A}$ such that with probability $1 - \epsilon$ w e have $\mathbb{E}(|\mathcal{L}_{\mathcal{A}} - \mathcal{L}_{\mathcal{A}^T}|) < 1 - \delta$.*

As a consequence of Theorem 4, there is no PAC-learning algorithm for LimAvg-automata with U-samples (resp., E-samples or (E, k)-samples) that uses samples of polynomial size; in particular, there is no such algorithm working in polynomial time.

▶ **Theorem 6.** *The class of* LimAvg*-automata is not PAC-learnable with U-samples, E-samples or (E, k)-samples.*

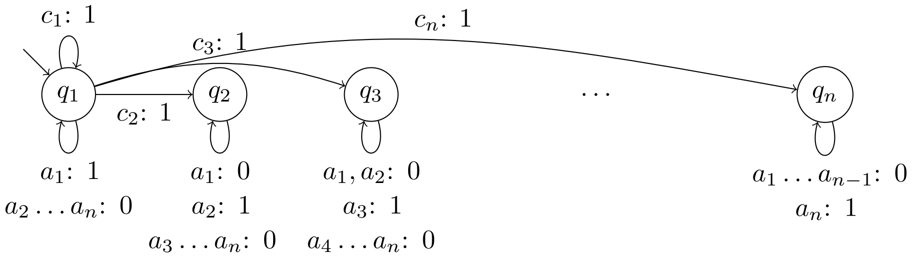


Figure 1 The canonical automaton $\mathcal{A}_\varphi$ from the proof of Theorem 8.

3.3 Sample fitting

Once we have a sample, the problem of finding an automaton fitting the sample can be solved in polynomial time in a trivial way: we create an automaton that is a tree such that every word of a given sample leads to a different leaf in this tree, and then we add loops with appropriate values in the leaves (similarly to a prefix tree acceptor [17] for finite automata). This solution leads to an automaton that *overfits* the samples, as it works well only for the sample and it is unlikely to work well with words not included in the sample. Besides, the automaton is linear in the size of the sample, not in the size of the black-box system. For a fixed automaton we can construct arbitrarily large U-samples (or E-samples) consistent with it and hence the gap between the size of such an automaton and the black-box system is arbitrarily large. To exclude such solutions, we restrict the size of the automaton to be constructed. We study the following problem.

▶ **Problem 7** (Sample fitting). *Given a sample S and $n \in \mathbb{N}$, construct a* LimAvg*-automaton with at most n states, which is consistent with S.*

The decision version of this problem only asks whether such an automaton exists. We show that this problem is NP-complete, regardless of the sample representation. For hardness, we reduce the NP-complete problem SAT_0 [17], which is the SAT problem restricted to CNF formulas such that each clause contains only positive literals or only negative literals.

▶ **Theorem 8.** *The sample fitting problem is* NP*-complete for U-samples.*

Proof. The membership in NP follows from the following observation: if n is greater than the total length of the samples, then return yes as the tree-like solution works. Otherwise, non-deterministically pick an automaton of the size n and check whether it fits the sample.

The NP-hardness proof is inspired by the construction from [17, Theorem 6.2.1]. For a given instance of the SAT_0 problem $\varphi = \bigwedge_{i=1}^{n} C_i$ over variables $x_1, \dots, x_n$ (not all variables need to occur in φ), we construct a U-sample S_φ such that there is an automaton with n states fitting S_φ if and only if φ is satisfiable.

We fix the alphabet $\{a_i, c_i, d_i \mid i = 1, \dots, n\} \cup \{b, t\}$. The sample S_φ consists of:

S1 (c_i, a_j, x) for each $i, j \in \{1, \dots, n\}$, where x is 1 if $i = j$, and 0 otherwise.

S2 (c_i, d_j, x) for each $i, j \in \{1, \dots, n\}$, where x is 1 if x_i is in C_j, and 0 otherwise.

S3 $(c_i b, d_i, 1)$ for each $i \in \{1, \dots, n\}$.

S4 $(c_i b, t, x)$ for each $i \in \{1, \dots, n\}$, where x is 1 if the clause C_i contains only positive literals and 0 if it contains only negative literals.

Assume that φ is satisfiable and let $\sigma : \{x_1, \dots, x_n\} \to \{0, 1\}$ be a satisfying valuation. Then, we construct an automaton $\mathcal{A}_\varphi$ consistent with the sample S_φ starting from the structure presented in Figure 1. Then, we add the following transitions:

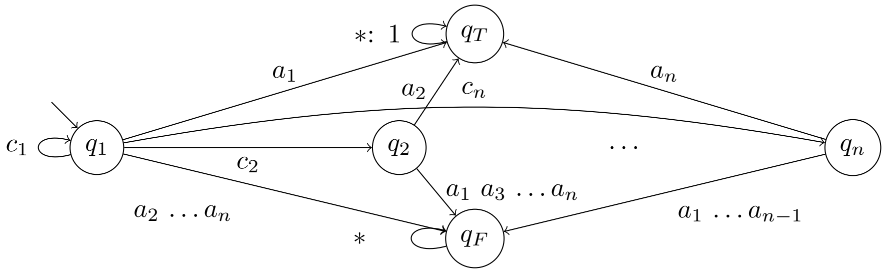


Figure 2 A picture of the canonical automaton from the proof of Theorem 9. All the weights are 0 except from transitions from the state q_T.

- for each i, a loop in q_i on the letter t with the value $\sigma(x_i)$,
- for each i, j, a loop in q_i on the letter d_j with the value 1 if x_i is in C_j and 0 otherwise,
- for each clause C_i, if C_i is satisfied because of a variable x_j, then we add a transition from q_i to q_j over b (if there are multiple possible variables, we choose any).

The remaining transitions can be set in arbitrary way. The obtained automaton $\mathcal{A}_\varphi$ is consistent with the sample S_φ.

Now assume that there is an automaton $\mathcal{A}$ of n states, which is consistent with the sample. We show that the valuation σ such that $\sigma(x_i) = \mathcal{L}_\mathcal{A}(c_i t^\omega)$ satisfies φ. Let q_i be the state where the automaton $\mathcal{A}$ is after reading the word c_i. By **S1**, all the states $q_1, \ldots, q_n$ are pairwise different. Since there are only n states, $q_1, \ldots, q_n$ are all the states of $\mathcal{A}$. Now consider any clause C_i. Let q_j be the state of $\mathcal{A}$ after reading $c_i b$. Notice that by **S3**, the value of d_i^ω in q_j is 1, and by **S2**, this means that x_j is in C_i. If C_i contains only positive literals, then the value of t^ω in q_j is 1 by **S4**, which means that $\sigma(x_j) = 1$ and that C_i is satisfied. The other case is symmetric. ◀

▶ **Theorem 9.** *The sample fitting problem is* NP*-complete for E-samples.*

Proof. The proof is similar to the proof of Theorem 8. For the NP-hardness, the sample now is obtained from the sample in the proof of Theorem 8 by replacing every triple (u, v, x) by the pair (uv, x). However, now we ask for an automaton of size $n + 2$. If there is a valuation that satisfies a given set of clauses, then one can construct an automaton based on the one presented in Figure 2.

On the other hand, if there is an automaton fitting the sample, then it has to have a state where the expected value of any word is 0, a state where the expected value of any word is 1, and n different states reachable by each $c_1, \ldots, c_n$. The rest of the proof is virtually the same as in Theorem 8, except that now we define σ such that $\sigma(x_i)$ is the expected value of words with the prefix $c_i t$. ◀

The above proofs also work with some natural relaxations of the sample fitting problem. For example, if we only require the automaton to fit the samples up to some $\epsilon < \frac{1}{2}$, then the proofs still hold since we use only weights 0 and 1. Another relaxation for the E-samples case is to allow the automaton to give wrong values for some samples as long as the summarized probability of the examples with wrong value is less than some ϵ. However, since all the words are of the length at most three, the probability of $u\Sigma^\omega$ for each example u is greater than $\frac{1}{(3n+2)^4}$ (recall that $|\Sigma| = 3n + 2$), which means that for any $\epsilon < \frac{1}{(3n+2)^4}$ for every example some its extension must fit and hence the whole sample must fit.

4 Active learning

In the active case, the learning algorithm can ask queries to an oracle, which is called *the teacher*, which has a (hidden) function $\mathcal{L} \colon \Sigma^\omega \to \mathbb{R}$ and answers two types of queries:

- *expectation queries*: given a finite word u, the teacher returns $\mathbb{E}(\mathcal{L} \mid u\Sigma^\omega)$,
- *ϵ-consistency queries*: given an automaton $\mathcal{A}$, if $\mathcal{L}_\mathcal{A}$ ϵ-approximates $\mathcal{L}$, the teacher answers YES, otherwise it returns a word u such that $|\mathbb{E}(\mathcal{L} \mid u\Sigma^\omega) - \mathbb{E}(\mathcal{L}_\mathcal{A} \mid u\Sigma^\omega)| > \epsilon$.

Observe that if $\mathcal{L}_\mathcal{A}$ does not ϵ-approximate $\mathcal{L}$, i.e., $\mathbb{E}(|\mathcal{L} - \mathcal{L}_\mathcal{A}|) > \epsilon$, a counterexample for the ϵ-consistency query exists. Indeed, if $\mathcal{L}_1, \mathcal{L}_2$ are defined by LimAvg-automata and $\mathbb{E}(|\mathcal{L}_1 - \mathcal{L}_2|) > \epsilon$, then there is a word $u\Sigma^\omega$ such that $\mathbb{E}(|\mathcal{L}_1 - \mathcal{L}_2| \mid u\Sigma^\omega) > \epsilon$ and $\mathcal{L}_1$ (resp., $\mathcal{L}_2$) returns $\mathbb{E}(\mathcal{L}_1 \mid u\Sigma^\omega)$ (resp,. $\mathbb{E}(\mathcal{L}_2 \mid u\Sigma^\omega)$) on almost all words from $u\Sigma^\omega$. Therefore, $|\mathbb{E}(\mathcal{L}_1 \mid u\Sigma^\omega) - \mathbb{E}(\mathcal{L}_2 \mid u\Sigma^\omega)| = \mathbb{E}(|\mathcal{L}_1 - \mathcal{L}_2| \mid u\Sigma^\omega) > \epsilon$.

In the active learning case we consider two problems: approximate learning and rigid approximate learning. We first define approximate learning:

▶ **Problem 10** (Approximate learning). *Given $\epsilon \in \mathbb{Q}^+ \cup \{0\}$ and a teacher with a (hidden) function $\mathcal{L}$, construct a* LimAvg*-automaton $\mathcal{A}$ such that $\mathcal{L}_\mathcal{A}$ ϵ-approximates $\mathcal{L}$ and $\mathcal{A}$ has the minimal number of state among such automata.*

4.1 Approximate learning

We define a decision problem, called approximate minimization, which can be solved in polynomial-time having a polynomial-time approximate learning algorithm.

▶ **Problem 11** (Approximate minimization). *Given a* LimAvg*-automaton $\mathcal{A}$, $n \in \mathbb{N}$ and $\epsilon \in \mathbb{Q}^+$*, the approximate minimization problem *asks whether there exists a* LimAvg*-automaton $\mathcal{A}'$ with at most n states such that $\mathbb{E}(|\mathcal{L}_\mathcal{A} - \mathcal{L}_{\mathcal{A}'}|) \leq \epsilon$.*

An efficient learning algorithm can be used to efficiently compute approximate minimization of a given LimAvg-automaton $\mathcal{A}$; we can run it and compute answers to queries of the learning algorithm in polynomial time in $|\mathcal{A}|$ [4]. We show that the approximate minimization problem is NP-complete, which means that approximate learning cannot be done in polynomial time if $P \neq NP$.

▶ **Theorem 12.** *The approximate minimization problem is* NP*-complete.*

Proof sketch. The problem is contained in NP as we can non-deterministically pick an automaton with n states and check whether it ϵ-approximates $\mathcal{A}$.

For a vector $\vec{v} \in \mathbb{R}^m$ we define $\|\vec{v}\|_1 = \sum_{i=1}^m |\vec{v}[i]|$. For NP-hardness, consider the following problem: *Binary k-Median Problem (BKMP)*: given numbers $n, m, k, t \in \mathbb{N}$ and a set of Boolean vectors $\mathcal{C} = \{\vec{v}_1, \ldots, \vec{v}_n\} \subseteq \{0,1\}^m$, decide whether there are vectors $\vec{u}_1, \ldots, \vec{u}_k \in \{0,1\}^m$ and a partition $\mathcal{D}_1, \ldots, \mathcal{D}_k$ of $\mathcal{C}$ such that $\sum_{j=1}^k \sum_{\vec{v} \in \mathcal{D}_j} \|\vec{v} - \vec{u}_j\|_1 \leq t$?

BKMP has been shown NP-complete in [30]. To ease the reduction, we consider a variant of BKMP, called Modified BKMP, where we assume that $\vec{0}, \vec{1}$ belong to the instance and the solution and some additional constraints. The Modified BKMP is also NP-complete.

For $\mathcal{C} = \{\vec{v}_1, \ldots, \vec{v}_n\} \subseteq \{0,1\}^m$ we define $\mathcal{L}_\mathcal{C}$ over the alphabet $\Sigma = \{a_1, \ldots, a_m\}$ as follows: for all $a_i, a_j \in \Sigma$, $w \in \Sigma^\omega$

- if $i \leq n$, we set $\mathcal{L}_\mathcal{C}(a_i a_j w) = \vec{v}_i[j]$.
- if $i \in \{n+1, m\}$, we set $\mathcal{L}_\mathcal{C}(a_i a_j w) = \vec{v}_n[j]$.

Intuitively, we select the vector with the first letter and the vector's component with the second letter. This language can be defined with a tree-like LimAvg-automaton $\mathcal{A}_{\mathcal{C}}$.

We show that, if an instance $(\mathcal{C}, k, t)$ of Modified BKMP has a solution $\vec{u}_1, \ldots, \vec{u}_k$ and $\mathcal{D}_1, \ldots, \mathcal{D}_k$, then there exists an automaton $\mathcal{A}$ with $k+1$ states that $\frac{t}{nm}$-approximates $\mathcal{L}_{\mathcal{C}}$. Let $\vec{v}_1 = \vec{u}_1 = \vec{0}$ and $\vec{v}_2 = \vec{u}_2 = \vec{1}$. The automaton $\mathcal{A}$ consists of the initial state q_0 and the successors of the initial state, $q_1, \ldots, q_k$, which correspond to vectors $\vec{u}_1, \ldots, \vec{u}_k$, i.e., q_1 is a bottom SCC of the value 0, q_2 is a bottom SCC of the value 1, and for $i > 2$ the successors of q_i encode $\vec{u}_i$ via $\delta(q_i, a_j) = q_1$ if $\vec{u}_i[j] = 0$ and $\delta(q_i, a_j) = q_2$ otherwise. The successors of q_0 are defined based on the partition $\mathcal{D}_1, \ldots, \mathcal{D}_k$, i.e., if $\vec{v}_i$ belongs to C_j, then $\delta(q_0, a_i) = q_j$. Observe that $\mathcal{A}$ ϵ-approximates $\mathcal{A}_{\mathcal{C}}$.

Conversely, consider $\mathcal{A}'$ with $k+1$ states that $\frac{t}{nm}$-approximates $\mathcal{L}_{\mathcal{C}}$. We define vectors $\vec{p}_1, \ldots, \vec{p}_m \in \mathbb{R}^m$ such that $\vec{p}_i[j] = \mathbb{E}(\mathcal{A}' \mid a_i a_j \Sigma^\omega)$. The structure of Modified BKMP, implies that the initial state of $\mathcal{A}'$ has no self loops and hence it has at most k different successor states. Therefore, there are at most k different vectors among $\vec{p}_1, \ldots, \vec{p}_m$. Finally, we observe that since we consider $\|\cdot\|_1$, w.l.o.g. we can assume that $\vec{p}_1, \ldots, \vec{p}_m \in \{0,1\}^m$. Therefore, these vectors give us a solution to the instance $(\mathcal{C}, k, t)$ of Modified BKMP. ◀

4.2 Rigid approximate learning

One of the drawbacks of the standard approximation is that the counterexamples may be dubious, if not useless. We illustrate this with an example.

▶ **Example 13** (Dubious counterexamples). Consider a minimal DFA $\mathcal{B}$ with 2^n states whose language consists of words of length m for some $n < m$, and a word $v \in \Sigma^n$. We define a function $\mathcal{L}_{\mathcal{B},v} \colon \{a,b\}^\omega \to \mathbb{R}$ such that for all u, w

- $\mathcal{L}_{\mathcal{B},v}(auw) = 0$ if $|u| = n$ and $\mathcal{B}$ accepts u,
- $\mathcal{L}_{\mathcal{B},v}(auw) = 0.3$ if $|u| = n$ and $\mathcal{B}$ rejects u,
- $\mathcal{L}_{\mathcal{B},v}(bw)$ is 0.4 if the first occurrence of v in w is followed by a, and 1 otherwise.

Fix $\epsilon = 0.1$. Observe that $\mathcal{L}_{\mathcal{B},v}$ can be 0.1-approximated with an automaton $\mathcal{A}$, which is faithful to $\mathcal{L}_{\mathcal{B},v}$ on $b\Sigma^\omega$ and returns 0.15 for all other words. $\mathcal{A}$ has $n + O(1)$ states.

Assume that the teacher gives only counterexamples starting with a, and hence ϵ-consistency queries do not give any information about the values of words starting with b. The teacher can do it as long as the algorithm does not know the whole $\mathcal{B}$, which takes $\Omega(|\mathcal{B}|)$ queries to learn. Yet even if the algorithm learns the whole $\mathcal{B}$ and returns the 0.7 for the words starting with b, the expected difference is $0.5 \cdot 0.3 = 0.15$. It follows that to learn the approximation, the algorithm needs to learn something about v.

Suppose that the algorithm did not learn the whole $\mathcal{B}$. Then, to learn something non-trivial about words starting with b, it has to ask an expectation query containing v. Since the learning algorithm is deterministic, it asks the same expectation queries for different words v. Therefore, for every learning algorithm there are words v that can be learned only after asking queries of the total length $2^{\Omega(|v|)}$.

It follows that any learning algorithm has to ask queries of total length $\Omega(|\mathcal{B}|)$ or $2^{\Omega(|v|)}$, which totals to $2^{\Omega(n)}$.

In Example 13 we assumed an antagonistic teacher, which misleads the algorithm on purpose. But even with a stochastic teacher, it is not known whether "fixing" a given random counterexample is a step towards better approximation. To resolve this issue, we consider a stronger notion of approximation, called *rigid approximation*, where we require all conditional expected values to be ϵ-close, i.e., for all words $u \in \Sigma^*$ we have $\mathbb{E}(|\mathcal{L}_1 - \mathcal{L}_2| \mid u\Sigma^\omega) \leq \epsilon$. In

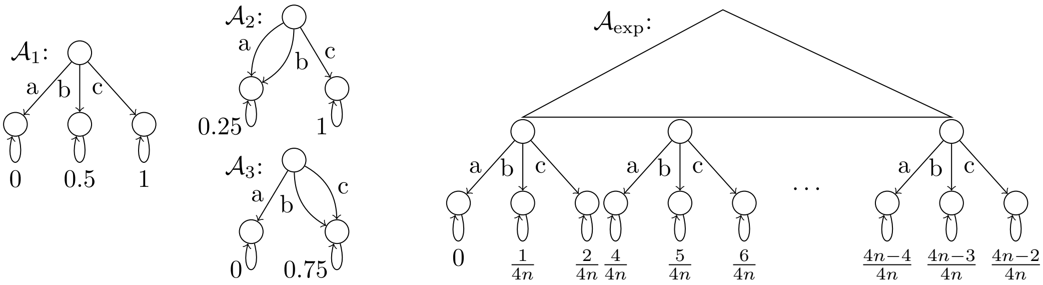


Figure 3 The automaton $\mathcal{A}_1$ approximated by two minimal non-equivalent automata and the automaton $\mathcal{A}_{\text{exp}}$ approximated by exponentially many non-equivalent automata.

this framework counterexamples are certain, i.e., if for some $u \in \Sigma^*$, the expectation over $u\Sigma^\omega$ is more than ϵ off the intended value, it has to be modified. Formally, we define the problem as follows:

▶ **Problem 14** (Rigid approximate learning). *Given $\epsilon \in \mathbb{Q}^+$ and a teacher with a (hidden) function $\mathcal{L}$, construct an automaton $\mathcal{A}$ such that $\mathcal{L}_\mathcal{A}$ is a rigid ϵ-approximation of $\mathcal{L}$ and $\mathcal{A}$ has the minimal number of state among such automata.*

Even though the counterexamples are certain in this framework, we observe that this does not eliminate ambiguity. For instance, there can be multiple automata with the minimal number of states.

▶ **Example 15** (Non-unique minimalization). Consider the automaton $\mathcal{A}_1$ depicted in Figure 3 and $\epsilon = \frac{1}{4}$. Any automaton $\mathcal{A}$, which is a rigid ϵ-approximation of $\mathcal{L}_{\mathcal{A}_1}$, has at least two bottom SCCs and hence it requires at least 3 states. Therefore the automata $\mathcal{A}_2, \mathcal{A}_3$ depicted in Figure 3, which ϵ-approximate $\mathcal{L}_{\mathcal{A}_1}$, have the minimal number of states. This shows that there can be multiple correct answers to in the rigid approximate learning problem.

Based on this example, we construct the automaton $\mathcal{A}_{\text{exp}}$ parametrized by $n \in \mathbb{N}$ depicted in Figure 3, which has $O(n)$ states and there are exponentially many (in n) minimal non-equivalent automata, which are rigid $\frac{1}{8n}$-approximations of $\mathcal{L}_{\mathcal{A}_{\text{exp}}}$.

As in the approximate learning case, efficient rigid approximate learning enables us to solve efficiently the following *rigid approximate minimization* problem.

▶ **Problem 16** (Rigid approximate minimization). *Given a* LimAvg*-automaton $\mathcal{A}$, $n \in \mathbb{N}$ and $\epsilon \in \mathbb{Q}^+$, construct a* LimAvg*-automaton $\mathcal{A}'$ with at most n states such that for all words $u \in \Sigma^*$ we have $\mathbb{E}(|\mathcal{L}_\mathcal{A} - \mathcal{L}_{\mathcal{A}'}| \mid u\Sigma^\omega) \leq \epsilon$.*

First, we consider a naive approach to solve the rigid approximate minimization problem based on state merging. We start with an input automaton $\mathcal{A}$, merge its states to maintain the property that the automaton with merged states $\mathcal{A}'$ rigidly ϵ-approximates $\mathcal{L}_\mathcal{A}$. We terminate if the automaton is *minimal*, i.e., merging any two states of $\mathcal{A}'$ violates the property. However, it may happen that q_1 can be merged either with q_2 or q_3, but not with both. We show in the following example that the choice of merged states can have profound impact on the size of a minimal automaton.

▶ **Example 17** (Minimal automata of different size). Assume $n \in \mathbb{N}$ and the function $\mathcal{L}$ that returns 0 on words from $a\Sigma^n a\Sigma^\omega$, 1 on $b\Sigma^n b\Sigma^\omega$, and 0.5 otherwise. Let $\mathcal{A}$ be a minimal automaton defining such a language, as depicted in Figure 4.

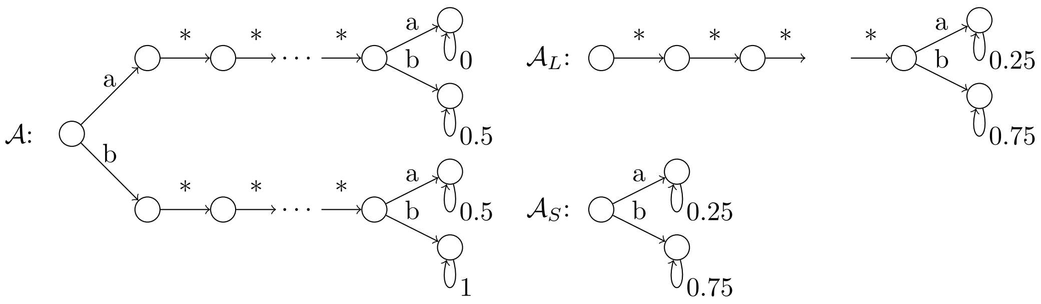


Figure 4 Minimal automata of different size.

Let $\epsilon = \frac{1}{4}$. To minimize $\mathcal{A}$, we can merge it to the automaton $\mathcal{A}_S$ with 3 states or to $\mathcal{A}_L$ with $n+3$ states. Observe that $\mathcal{A}_S$ and $\mathcal{A}_L$ are rigid ϵ-approximations of $\mathcal{L}_{\mathcal{A}}$. For $\mathcal{A}_S$, it is because for every word au we have $\mathbb{E}(\mathcal{A}_B \mid au\Sigma^\omega) \in \{0, 0.5\}$ and hence it is ϵ-close to 0.25, and similarly for ba. For $\mathcal{A}_L$, it is because for every u of length at least $n+2$ the expected value of $\mathcal{A}_L$ is 0.75 or 0.25 depending on the $(n+2)$th letter of u whereas for $\mathcal{A}$ it is in $\{0.5, 1\}$ or in $\{0, 0.5\}$, resp. For shorter u, we simply observe that the difference of expected values w.r.t. to a word does not exceed the maximal difference in its suffixes.

The automaton $\mathcal{A}_L$ is minimal, as there are no states that can be merged. Therefore, the difference and even the ratio between the sizes of both automata are unbounded.

We show that the rigid approximate minimization problem is NP-complete, which implies that there is no polynomial-time rigid approximate learning algorithm (unless $P = NP$).

▶ **Theorem 18.** *The rigid approximate minimization problem is* NP*-complete.*

Proof sketch. The rigid approximate minimization problem is in NP. We show that it is NP-hard. We define a problem, which is an intermediate step in our reduction.

Given $n, k \in \mathbb{N}$ and vectors $\mathcal{C} = \{\vec{v}_1, \ldots, \vec{v}_n\} \subseteq \{0, \frac{1}{2}, 1\}^n$, *the* $\frac{1}{4}$*-vector cover problem* asks whether there exist $\vec{u}_1, \ldots, \vec{u}_k \in \mathbb{R}^n$ such that for every $\vec{v} \in \mathcal{C}$ there is $\vec{u}_j$ with $\|\vec{v} - \vec{u}_j\|_\infty \leq \frac{1}{4}$?

The $\frac{1}{4}$-vector cover problem is NP-complete; the NP-hardness is via reduction from the dominating set problem. We show that the $\frac{1}{4}$-vector cover problem reduces to the rigid approximate minimization problem

Let $\mathcal{C} = \{\vec{v}_1, \ldots, \vec{v}_n\} \subseteq \{0, \frac{1}{2}, 1\}^n$. We define $\mathcal{L}_{\mathcal{C}}$ over the alphabet $\Sigma = \{a_1, \ldots, a_m\}$ such that for all $a_i, a_j \in \Sigma$, $w \in \Sigma^\omega$ we have $\mathcal{L}_{\mathcal{C}}(a_i a_j w) = \vec{v}_i[j]$. Such $\mathcal{L}_{\mathcal{C}}$ can be defined with a a tree-like LimAvg-automaton $\mathcal{A}_{\mathcal{C}}$. We show that an instance $n, k, \mathcal{C}$ of the $\frac{1}{4}$-vector cover problem has a solution if and only if $\mathcal{A}_{\mathcal{C}}$ has a rigid $\frac{1}{4}$-approximation with $k+3$ states.

Assume that $\mathcal{A}'$ is an automaton with $k+3$ states such that for all words $u \in \Sigma^*$ we have $\mathbb{E}(|\mathcal{L}_{\mathcal{A}_\mathcal{C}} - \mathcal{L}_{\mathcal{A}'}| \mid u\Sigma^\omega) \leq \frac{1}{4}$. Let p_0 be initial in $\mathcal{A}$; we show that it has at most k different successors because at least 3 states cannot be successors of p_0: p_0 itself, and p_1, p_2 being states in two different bottom SSCs (that need to exist). Otherwise, if u leads to p_i $(i = 0, 1, 2)$ we have $|\mathbb{E}(|\mathcal{L}_{\mathcal{C}}(w) - \mathcal{L}_{\mathcal{C}}(uw)| \mid w \in \Sigma^\omega) > \frac{1}{2}$, which contradicts $\mathcal{A}'$ being $\frac{1}{4}$-approximation. Thus, p_0 has at most k different successors; We define vectors $\vec{u}_1, \ldots, \vec{u}_k$ from the successors of p_0. Formally, for $i \in \{1, \ldots, n\}$ we define a vector $\vec{y}_i$ as $\vec{y}_i[j] = \mathbb{E}(\mathcal{L}_{\mathcal{A}'} \mid a_i a_j \Sigma^\omega)$. There are at most k different vectors among $\vec{y}_1, \ldots, \vec{y}_n$ and we take these distinct vectors as $\vec{u}_1, \ldots, \vec{u}_k$. The condition $\mathbb{E}(|\mathcal{L}_{\mathcal{A}_\mathcal{C}} - \mathcal{L}_{\mathcal{A}'}| \mid u\Sigma^\omega) \leq \frac{1}{4}$ implies that $\vec{u}_1, \ldots, \vec{u}_k$ form a solution to $n, k, \mathcal{C}$.

Conversely, assume that the instance $n, k, \mathcal{C}$ has a solution. Observe that w.l.o.g. we can assume that the solution vectors $\vec{u}_1, \ldots, \vec{u}_k$ belong to $\{0.25, 0.75\}^n$. Based on this solution we define $\mathcal{A}'$ with $k+3$ states $q_0, q_1, \ldots, q_k, s_1, s_2$ such that q_0 is initial, $q_1, \ldots, q_k$ are successors of q_0, and s_1, s_2 are single-state bottom SCC of the values 0.25 and 0.75. The states $q_1, \ldots, q_k$ correspond to vectors $\vec{u}_1, \ldots, \vec{u}_k$, i.e., the successors of q_i encode $\vec{u}_i$ via $\delta(q_i, a_j) = s_1$ if $\vec{u}_i[j] = 0.25$ and $\delta(q_i, a_j) = s_2$ otherwise. The successors of q_0 are defined based on the matching from the vector cover, i.e., if $\delta(q_0, a_i) = q_j$, then $\vec{u}_j$ is $\frac{1}{4}$-close to $\vec{v}_i$. Observe that $\mathcal{A}'$ is a rigid $\frac{1}{4}$-approximation of $\mathcal{A}_{\mathcal{C}}$. ◀

5 Almost-exact learning

The almost-exact learning the minimal automaton is defined as follows.

▶ **Problem 19** (Almost-exact learning). *Given a teacher with a (hidden) function $\mathcal{L}$, construct a* LimAvg*-automaton $\mathcal{A}$ such that $\mathcal{L}_{\mathcal{A}}$ is a 0-approximation of $\mathcal{L}$.*

Notice that for functions $\mathcal{L}$ and $\mathcal{L}_{\mathcal{A}}$, the following conditions are equivalent:

- $\mathcal{L}_{\mathcal{A}}$ is a 0-approximation of $\mathcal{L}$.
- $\mathcal{L}_{\mathcal{A}}$ is a rigid 0-approximation of $\mathcal{L}$.
- $\mathbb{P}(\{w : \mathcal{L}_{\mathcal{A}}(w) \neq \mathcal{L}(w)\})$=0.
- $\mathbb{P}(\{w : \mathcal{L}_{\mathcal{A}}(uw) \neq \mathcal{L}(uw)\})$=0 for each u.

We show that there is a polynomial-time algorithm for almost-exact learning.

▶ **Theorem 20.** *The almost-exact learning problem for* LimAvg*-automata can be solved in polynomial time in the size of the minimal automaton that is almost equivalent with the target function $\mathcal{L}$ and the maximal length of counterexamples returned by the teacher.*

Proof. We define a relation $\equiv^0_{\mathcal{L}}$ on finite words $u, v \in \Sigma^*$ as follows:

$$u \equiv^0_{\mathcal{L}} v \text{ if and only if } \mathbb{P}(\{w : \mathcal{L}(uw) \neq \mathcal{L}(vw)\}) = 0$$

Clearly, $\equiv^0_{\mathcal{L}}$ is an equivalence relation. We show that $\equiv^0_{\mathcal{L}}$ is a right congruence, i.e., if $u \equiv^0_{\mathcal{L}} v$ and $a \in \Sigma$, then $ua \equiv^0_{\mathcal{L}} va$. Indeed, consider $X_1 = \{w : \mathcal{L}(uaw) \neq \mathcal{L}(vaw)\}$ and $X_2 = \{w : \mathcal{L}(uw) \neq \mathcal{L}(vw)\}$. Note that $u \equiv^0_{\mathcal{L}} v$ implies $\mathbb{P}(X_2) = 0$. For all w, if $w \in X_1$, then $aw \in X_2$. Thus, under the uniform distribution we have $\mathbb{P}(X_1) \leq \frac{1}{|\Sigma|}\mathbb{P}(X_2) = 0$ and hence $\mathbb{P}(X_1) = 0$ and $ua \equiv^0_{\mathcal{L}} va$.

We now show a counterpart of the Myhill–Nerode theorem: there is a LimAvg-automaton of n states defining almost-exactly $\mathcal{L}$ if and only if the index of $\equiv^0_{\mathcal{L}}$ is n.

If $\mathcal{A}$ defines almost-exactly $\mathcal{L}$, then the index of $\equiv^0_{\mathcal{L}}$ is bounded by the number of states of $\mathcal{A}$. Indeed, if for words u, v the automaton $\mathcal{A}$ starting from the initial state ends up in the same state, then for all words w we have $\mathcal{L}(uw) = \mathcal{L}(vw)$ and hence $u \equiv^0_{\mathcal{L}} v$. Conversely, assume that $\equiv^0_{\mathcal{L}}$ has a finite index. Then, we construct a LimAvg-automaton $\mathcal{A}_{\equiv^0_{\mathcal{L}}}$ corresponding to $\equiv^0_{\mathcal{L}}$. The states of $\mathcal{A}_{\equiv^0_{\mathcal{L}}}$ are the equivalence classes of $\equiv^0_{\mathcal{L}}$ and $\mathcal{A}_{\equiv^0_{\mathcal{L}}}$ has a transition from $[u]_{\equiv^0_{\mathcal{L}}}$ to $[v]_{\equiv^0_{\mathcal{L}}}$ over a if and only if $[ua]_{\equiv^0_{\mathcal{L}}} = [v]_{\equiv^0_{\mathcal{L}}}$. Observe that due to Theorem 1, if $[u]_{\equiv^0_{\mathcal{L}}}$ and $[v]_{\equiv^0_{\mathcal{L}}}$ are in a bottom SCC, then $[u]_{\equiv^0_{\mathcal{L}}} = [v]_{\equiv^0_{\mathcal{L}}}$. Then, for every bottom SCCs $[u]_{\equiv^0_{\mathcal{L}}}$ we assign the value of all outgoing transitions (which are self-loops) to $\mathbb{E}(\mathcal{L} \mid u\Sigma^\omega)$. For the remaining transitions we set the weights to 0 (due to Theorem 1 these weights are irrelevant as they do not change any of the expected values). Observe that $\mathcal{A}_{\equiv^0_{\mathcal{L}}}$ computes $\mathcal{L}$.

A classical result of [1] states that DFA can be learned in polynomial time using membership and equivalence queries. We adapt this result here. The learning algorithm for LimAvg-automata maintains a pair $(\mathcal{Q}, \mathcal{T})$, where $\mathcal{Q}$, the set of access words, contains different representatives the right congruence relation, and $\mathcal{T}$, the set of test words, contains words that approximate the right congruence relation. $\mathcal{T}$ defines the relation $\equiv_{\mathcal{T}}$ such that $u_1 \equiv_{\mathcal{T}} u_2$ if and only if for all $v \in \mathcal{T}$ we have $\mathbb{P}(\{w \mid \mathcal{L}(u_1vw) \neq \mathcal{L}(u_2vw)\}) = 0$.

The algorithm maintains two properties: *separability*: all different words $u_1, u_2 \in \mathcal{Q}$ belong to different equivalence classes of $\equiv_{\mathcal{T}}$, and *closedness*; for all $u_1 \in \mathcal{Q}$ and $a \in \Sigma$ there is $u_2 \in \mathcal{Q}$ with $u_1 a \equiv_{\mathcal{T}} u_2$. Each separable and closed pair $(\mathcal{Q}, \mathcal{T})$ defines a LimAvg-automaton $\mathcal{A}_{\mathcal{Q},\mathcal{T}}$, which can be tested for 0-consistency against the teacher's function. If the teacher provides a counterexample u, then it is used to extend $\mathcal{Q}$ and $\mathcal{T}$. To do so, we split u into $v_1 a, v_2$ such that $v_1 a$ is the minimal prefix of u such that $\mathbb{E}(\mathcal{L} \mid v_1 \Sigma^\omega) \neq \mathbb{E}(\mathcal{A}_{\mathcal{Q},\mathcal{T}} \mid u\Sigma^\omega)$. Let v_1' be a word from $\mathcal{Q}$ that is $\equiv_{\mathcal{T}}$-equivalent to v_1. We take $\mathcal{Q}' = \mathcal{Q} \cup \{v_1' a\}, \mathcal{T}' = \mathcal{T} \cup \{v_2\}$ and close $(\mathcal{Q}', \mathcal{T}')$ using expectation queries and test the equivalence again. We repeat this until we get a LimAvg-automaton defining almost equivalent to $\mathcal{L}$.

Correctness of this algorithm follows from correctness of the algorithm from [1]. ◀

Non-uniform distributions. So far we only discussed the uniform distribution of words. Here we briefly discuss whether Theorem 20 can be generalized to arbitrary distributions, represented by Markov chains. We assume that the Markov chain is given to a learning algorithm in the input.

A (finite-state discrete-time) *Markov chain* is a tuple $\langle \Sigma, S, s_0, E \rangle$, where Σ is the alphabet of letters, S is a finite set of states, s_0 is an initial state, $E \colon S \times \Sigma \times S \mapsto [0,1]$ is an edge probability function, which for every $s \in S$ satisfies $\sum_{a \in \Sigma, s' \in S} E(s, a, s') = 1$.

The probability of a finite word u w.r.t. a Markov chain $\mathcal{M}$, denoted by $\mathbb{P}_{\mathcal{M}}(u)$, is the sum of probabilities of paths from s_0 labeled by u, where the probability of a path is the product of probabilities of its edges. For basic open sets $u \cdot \Sigma^\omega = \{uw \mid w \in \Sigma^\omega\}$, we have $\mathbb{P}_{\mathcal{M}}(u \cdot \Sigma^\omega) = \mathbb{P}_{\mathcal{M}}(u)$, and then the probability measure over infinite words defined by $\mathcal{M}$ is the unique extension of the above measure [19]. We will denote the unique probability measure defined by $\mathcal{M}$ as $\mathbb{P}_{\mathcal{M}}$. The uniform distribution can be expressed with a (single-state) Markov chain and hence all the lower bounds from Section 3 and Section 4 apply here.

A Markov chain $\mathcal{M}$ is *non-vanishing* if for all words $u \in \Sigma^*$ we have $\mathbb{P}_{\mathcal{M}}(u\Sigma^*) > 0$. The almost-exact learning over distributions given by non-vanishing Markov chains and over $\mathcal{M}$ and over the uniform distribution coincide. Thus, the former can be solved using Theorem 20. Indeed, if X is measurable and $\mathcal{M}$ is non-vanishing, then $\mathbb{P}_{\mathcal{M}}(X) > 0$ if and only if $\mathbb{P}(X) > 0$.

The algorithm for the uniform distribution does not extend to vanishing Markov chains because the relation $\equiv^0_{\mathcal{L}}$ is not a right congruence. This cannot be simply fixed as we show that learning cannot be done in polynomial time, assuming P $\neq$ NP. We define the almost-exact minimization problem as an instance of the approximate minimization problem with $\epsilon = 0$. Having a polynomial-time algorithm for almost-exact learning, we can solve the almost-exact minimization problem in polynomial time.

▶ **Theorem 21.** *The almost-exact minimization problem for* LimAvg*-automata under distributions given by Markov chains is* NP*-complete.*

Proof. The problem is in NP: we can non-deterministically pick an automaton $\mathcal{A}'$ and check in polynomial time whether $\mathcal{A}$ and $\mathcal{A}'$ are almost equivalent w.r.t. a given Markov chain.

We reduce the sample fitting problem for E-samples, which is NP-complete (Theorem 9), to the almost-exact minimization problem. Consider an E-sample S based on words $u_1, \ldots, u_k$. Let $\mathcal{M}_S$ be a Markov chain which assigns probability $\frac{1}{k}$ to $u_i \Sigma^\omega$, for each i, and 0 to words not starting with any of u_i. On each $u_i \Sigma^\omega$, $\mathcal{M}_S$ defines the uniform distribution. Let $\mathcal{A}_S$ be a a tree-like LimAvg-automaton consistent with S. Both, $\mathcal{M}_S$ and $\mathcal{A}_S$ are of polynomial size in S. Observe that every automaton $\mathcal{A}$, which is consistent with S (according to the uniform distribution over infinite words), is almost equivalent to $\mathcal{A}_S$ (over $\mathbb{P}_{\mathcal{M}}$) and vice versa. Therefore, there is an automaton of n states almost equivalent to $\mathcal{A}_S$ (under the distribution $\mathbb{P}_{\mathcal{M}}$) if and only if the sample fitting problem with S and n has a solution. ◀

References

1 Dana Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

2 Dana Angluin and Dongqu Chen. Learning a Random DFA from Uniform Strings and State Information. In *ALT 2015*, pages 119–133, Cham, 2015. Springer International Publishing.

3 Dana Angluin and Dana Fisman. Learning regular omega languages. *Theor. Comput. Sci.*, 650:57–72, 2016.

4 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

5 Borja Balle and Mehryar Mohri. Learning Weighted Automata. In *CAI 2015*, pages 1–21, 2015.

6 Borja Balle and Mehryar Mohri. On the Rademacher Complexity of Weighted Automata. In *ALT 2015*, pages 179–193, 2015.

7 Borja Balle and Mehryar Mohri. Generalization bounds for learning weighted automata. *Theor. Comput. Sci.*, 716:89–106, 2018.

8 Borja Balle, Prakash Panangaden, and Doina Precup. A Canonical Form for Weighted Automata and Applications to Approximate Minimization. In *LICS 2015*, pages 701–712, 2015.

9 Amos Beimel, Francesco Bergadano, Nader Bshouty, Eyal Kushilevitz, and Stefano Varricchio. Learning Functions Represented as Multiplicity Automata. *Journal of the ACM*, 47, October 1999.

10 Michael Benedikt, Gabriele Puppis, and Cristian Riveros. Regular Repair of Specifications. In *LICS 2011*, pages 335–344, 2011.

11 Udi Boker, Krishnendu Chatterjee, Thomas A. Henzinger, and Orna Kupferman. Temporal Specifications with Accumulative Values. *ACM Trans. Comput. Log.*, 15(4):27:1–27:25, 2014.

12 Benedikt Bollig, Peter Habermehl, Carsten Kern, and Martin Leucker. Angluin-Style Learning of NFA. In *IJCAI 2009*, pages 1004–1009, 2009.

13 Patricia Bouyer, Nicolas Markey, and Raj Mohan Matteplackel. Averaging in LTL. In *CONCUR 2014*, pages 266–280, 2014.

14 Pavol Cerný, Thomas A. Henzinger, and Arjun Radhakrishna. Quantitative abstraction refinement. In *POPL 2013*, pages 115–128, 2013.

15 Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. Quantitative languages. *ACM TOCL*, 11(4):23, 2010.

16 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Quantitative Automata under Probabilistic Semantics. In *LICS 2016*, pages 76–85, 2016.

17 Colin de la Higuera. *Grammatical Inference: Learning Automata and Grammars.* Cambridge University Press, New York, NY, USA, 2010.

18 Manfred Droste, Werner Kuich, and Heiko Vogler. *Handbook of Weighted Automata.* Springer, 1st edition, 2009.

19 W. Feller. *An introduction to probability theory and its applications.* Wiley, 1971.

20 Dana Fisman. Inferring regular languages and ω-languages. *J. Log. Algebr. Meth. Program.*, 98:27–49, 2018.

21 E Mark Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.

22 Amaury Habrard and José Oncina. Learning Multiplicity Tree Automata. In *ICGI 2006*, pages 268–280, 2006.

23 Thomas A. Henzinger and Jan Otop. Model measuring for discrete and hybrid systems. *Nonlinear Analysis: Hybrid Systems*, 23:166–190, 2017.

24 Jeffrey C Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *Journal of Computer and System Sciences*, 55(3):414–440, 1997.

25 Michael Kearns and Leslie Valiant. Cryptographic limitations on learning Boolean formulae and finite automata. *Journal of the ACM (JACM)*, 41(1):67–95, 1994.

26 Michael J Kearns, Umesh Virkumar Vazirani, and Umesh Vazirani. *An introduction to computational learning theory.* MIT Press, 1994.

27 Kevin J. Lang. Random DFA's Can Be Approximately Learned from Sparse Uniform Examples. In *COLT 1992*, pages 45–52, 1992.

28 Ines Marusic and James Worrell. Complexity of equivalence and learning for multiplicity tree automata. *Journal of Machine Learning Research*, 16:2465–2500, 2015.

29 Jakub Michaliszyn and Jan Otop. Non-deterministic Weighted Automata on Random Words. In *CONCUR 2018*, volume 118 of *LIPIcs*, pages 10:1–10:16, 2018.

30 Pauli Miettinen, Taneli Mielikäinen, Aristides Gionis, Gautam Das, and Heikki Mannila. The discrete basis problem. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 335–346. Springer, 2006.

31 Joshua Moerman, Matteo Sammartino, Alexandra Silva, Bartek Klin, and Michal Szynwelski. Learning nominal automata. In *POPL 2017*, pages 613–625, 2017.

32 José Oncina and Pedro Garcia. Inferring regular languages in polynomial updated time. In *Pattern recognition and image analysis: selected papers from the IVth Spanish Symposium*, pages 49–61. World Scientific, 1992.

33 Leonard Pitt. Inductive inference, DFAs, and computational complexity. In *International Workshop on Analogical and Inductive Inference*, pages 18–44. Springer, 1989.

34 Leonard Pitt and Manfred K Warmuth. The minimum consistent DFA problem cannot be approximated within any polynomial. *Journal of the ACM (JACM)*, 40(1):95–142, 1993.

35 Leslie G Valiant. A theory of the learnable. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 436–445. ACM, 1984.

Alternating Weak Automata from Universal Trees

Laure Daviaud
City, University of London, UK
Laure.Daviaud@city.ac.uk

Marcin Jurdziński
University of Warwick, UK
Marcin.Jurdzinski@warwick.ac.uk

Karoliina Lehtinen
University of Liverpool, UK
k.lehtinen@liverpool.ac.uk

Abstract

An improved translation from alternating parity automata on infinite words to alternating weak automata is given. The blow-up of the number of states is related to the size of the smallest universal ordered trees and hence it is quasi-polynomial, and it is polynomial if the asymptotic number of priorities is at most logarithmic in the number of states. This is an exponential improvement on the translation of Kupferman and Vardi (2001) and a quasi-polynomial improvement on the translation of Boker and Lehtinen (2018). Any slightly better such translation would (if – like all presently known such translations – it is efficiently constructive) lead to algorithms for solving parity games that are asymptotically faster in the worst case than the current state of the art (Calude, Jain, Khoussainov, Li, and Stephan, 2017; Jurdziński and Lazić, 2017; and Fearnley, Jain, Schewe, Stephan, and Wojtczak, 2017), and hence it would yield a significant breakthrough.

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

Keywords and phrases alternating automata, weak automata, Büchi automata, parity automata, parity games, universal trees

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.18

Funding This work has been supported by the EPSRC grants EP/P020992/1 and EP/P020909/1 (Solving Parity Games in Theory and Practice).

Acknowledgements We thank Moshe Vardi for encouraging us to bring the state-space blow-up of alternating parity to alternating weak automata translation in line with the state-of-the-art complexity of solving parity games.

1 Introduction

The influential class of regular languages of infinite words (often called the ω-regular languages) is defined to consist of all the languages of infinite words that are recognized by finite non-deterministic Büchi automata. The theory of ω-regular languages is quite well understood. In particular, it is known that deterministic Büchi automata are not sufficiently expressive to recognize all the ω-regular languages, but deterministic automata with the so-called parity acceptance conditions are, and that the class of ω-regular languages is closed under complementation. Effective constructions for determinization and complementation of Büchi automata are important tools both in theory and in applications, and both are known to require exponential blow-ups of the numbers of states in the worst case.

For applications in logic, it is natural to enrich automata models by the ability to alternate between non-deterministic and universal transitions [21, 16]. It turns out that alternating parity automata are no more expressive than non-deterministic Büchi automata,

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 18; pp. 18:1–18:14
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and hence neither allowing alternation, nor the richer parity acceptance conditions, increase expressiveness; this testifies to the robustness of the class of ω-regular languages. On the other hand, alternation increases the expressive power of automata with the so-called weak acceptance conditions: non-deterministic weak automata are not expressive enough to recognize all ω-regular languages, but alternating weak automata are. The weak acceptance conditions are significant due to their applications in logic [21] and thanks to their favourable algorithmic properties [16].

Given that alternating weak automata are expressive enough to recognize all the ω-regular languages, a natural question is whether, and to what degree, alternating weak automata are less succinct than alternating Büchi or alternating parity automata. Another way of stating this question is what blow-up in the number of states is sufficient or required for translations from alternating parity or alternating Büchi automata to alternating weak automata. The first upper bound for the blow-up of a translation from alternating Büchi to alternating weak automata was doubly exponential, obtained by combining a doubly-exponential determinization construction [7] and a linear translation from deterministic parity automata to weak alternating automata [21, 19]. This has been improved considerably by Kupferman and Vardi who have given a quadratic translation from alternating Büchi to alternating weak automata [15], and then they have generalized it to a translation from alternating parity automata with n states and d priorities to alternating weak automata, whose blow-up is $n^{d+O(1)}$, i.e., exponential in the number of priorities in the parity acceptance condition [14].

Understanding the exact trade-off between the complexity of the acceptance condition – weak, Büchi, or parity, the latter measured by the number of priorities – and the number of states in an automaton is interesting from the algorithmic point of view. For example, the algorithmic problems of checking emptiness of non-deterministic parity automata on infinite trees, of model checking for the modal μ-calculus, of solving two-player parity games, and of checking emptiness of alternating parity automata on infinite words over a one-letter alphabet, are all polynomial-time equivalent. Since checking emptiness of alternating weak automata on words over a one-letter alphabet can be done in linear time, it follows that a translation from alternating parity automata to alternating weak automata implies an algorithm for solving parity games whose complexity matches the blow-up of the number of states in the translation.

The first quasi-polynomial translation from alternating parity automata to alternating weak automata was given recently by Boker and Lehtinen [1]. They have used the register technique, developed by Lehtinen [17] for parity games, to provide a translation from alternating parity automata with n states and d priorities to alternating parity automata with $n^{\Theta(\log(d/\log n))}$ states and $\Theta(\log n)$ priorities; combined with the exponential translation of Kupferman and Vardi [14], this yields an alternating parity to alternating weak translation whose blow-up of the number of states is $n^{\Theta(\log n \cdot \log(d/\log n))}$.

The main result reported in this paper is that another technique – universal trees [11, 5], also developed to elucidate the recent major advance in the complexity of solving parity games due to Calude, Jain, Khoussainov, Li, and Stephan [2] – can be used to further reduce the state-space blow-up in the translation from alternating parity automata to alternating weak automata. We give a translation from alternating parity automata with n states and d priorities to alternating Büchi automata, whose state-space blow-up is proportional to the size of the smallest $(n, d/2)$-universal trees [5], which is polynomial in n if $d = O(\log n)$ and it is $n^{\lg(d/\lg n)+O(1)}$ if $d = \omega(\log n)$. When combined with Kupferman and Vardi's quadratic translation of alternating Büchi to alternating weak automata [15], we get the composite blow-up of the form $n^{O(\log(d/\log n))}$, down from Boker and Lehtinen's blow-up of $\left(n^{\Theta(\log(d/\log n))}\right)^{\log n}$.

The necessary size of the state-space blow-up when going from alternating parity automata to alternating weak automata is wide open: the best known lower bound is $\Omega(n \log n)$ [15], closely related to the $2^{\Omega(n \log n)}$ lower bound on Büchi complementation [20], while the best upper bounds are quasi-polynomial. On the other hand, the blow-up for the parity to weak translation that we obtain nearly matches the current state-of-the-art quasi-polynomial upper bounds on the complexity of solving parity games [2, 11, 9]. It follows that any significant improvement over our translation would lead to a breakthrough improvement in the complexity of solving parity games.

The exponential translation from alternating parity automata to alternating weak automata due to Kupferman and Vardi [14] is done by a rather involved induction on the number of priorities. For an automaton with d priorities, it goes through a sequence of d intermediate automata of a generalized type, which they call parity-weak alternating automata. In contrast, our construction is significantly more streamlined and transparent; in particular, it avoids introducing a new class of hybrid parity-weak automata. We first establish a hierarchical decomposition of runs of alternating parity automata as a generalization of the decomposition of runs of alternating co-Büchi automata due to Kupferman and Vardi [15], and then we use the recently introduced universal trees [11, 5] to construct an alternating Büchi automaton, which is a parity automaton with just 2 priorities. Our work is yet another application of the recently introduced notion of universal trees [11, 5]. Such applications typically focus on algorithms for solving games [11, 6, 5, 3]; our work is the first whose primary focus is on automata.

In addition to universal trees, we use a notion of lazy progress measure. Unlike the standard parity progress measures, which can be recognised by safety automata but require an explicit bound to be known on the number of successive occurrences of odd priorities, lazy progress measures are recognised by Büchi automata and can deal with finite but unbounded numbers of occurrences of successive odd priorities. This Büchi automaton is similar to (but more concise than) the automaton used to characterise parity tree automata that recognise co-Büchi recognisable tree languages [18], itself a generalisation of automata used to decide the weak definability of tree languages given as Büchi automata [22, 4].

A similar concept to our lazy progress measures was already introduced by Klarlund for complementation of Büchi and Streett automata on words [12]. Klarlund indeed proves a result that is equivalent to one of our key lemmas on parity progress measures. Our proof, however, is more constructive, and it explicitly provides a hierarchical decomposition, which clearly describes the structure of accepting run dags of parity word automata. Moreover – unlike Klarlund's proof, which relies on the result about Rabin measures [13] – our proof is self-contained. We suspect that the opaqueness of Klarlund's paper [12] may have been responsible for attracting less attention and shallower absorption by the research community than it deserves. In particular, some of the techniques and results that he presents there have been rediscovered and refined by various authors, often much later [15, 14, 10, 11, 5], including this work. We hope that our paper will help a wider and more thorough reception and appreciation of Klarlund's work.

2 Alternating automata

For a finite set X, we write $\mathcal{B}^+(X)$ for the set of positive Boolean formulas over X. We say that a set $Y \subseteq X$ *satisfies* a formula $\varphi \in \mathcal{B}^+(X)$ if φ evaluates to `true` when all variables in Y are set to `true` and all variables in $X \setminus Y$ are set to `false`. For example, the sets $\{x, y\}$ and $\{x, z\}$ satisfy the positive Boolean formula $x \wedge (y \vee z)$, but the set $\{y, z\}$ does not. An *alternating automaton* has a finite set Q of *states*, an *initial state* $q_0 \in Q$, a finite *alphabet* Σ, and a transition function $\delta : Q \times \Sigma \to \mathcal{B}^+(Q)$. Alternating automata allow to combine both *non-deterministic* and *universal* transitions; disjunctions in transition formulas model the non-deterministic choices and conjunctions model the universal choices.

We consider alternating automata as acceptors of infinite words. Whether infinite sequences of states in runs of such automata are *accepting* or not is determined by an *acceptance condition.* Here, we consider *parity*, *Büchi*, *co-Büchi*, *weak*, and *safety* acceptance conditions. In a *parity* condition, given by a *state priority function* $\pi : Q \to \{0, 1, 2, \dots, d\}$ for some positive even integer d, an infinite sequence of states is accepting if the largest state priority that occurs infinitely many times is even. *Büchi* conditions are a special case of parity conditions in which all states have priorities 1 or 2, and *co-Büchi* conditions are parity conditions in which all states have priorities 0 or 1.

Let the *transition graph* of an alternating automaton have an edge $(q, r) \in Q \times Q$ if r occurs in $\delta(q, a)$ for some letter $a \in \Sigma$. We say that a parity automaton has a *weak* acceptance condition if it is *stratified*: in every cycle in the transition graph, all states have the same priority. Weak conditions are a special case of both Büchi and co-Büchi conditions in the following sense: if the transition graph of a parity automaton is stratified, then every infinite path in the transition graph satisfies each of the following three conditions if and only if it satisfies the other two:

- the parity condition $\pi : Q \to \{0, 1, 2, \dots, d\}$;
- the co-Büchi condition $\pi' : Q \to \{0, 1\}$;
- the Büchi condition $\pi'' : Q \to \{1, 2\}$;

where $\pi'(q) = \pi(q) \bmod 2$ and $\pi''(q) = 2 - \pi'(q)$ for all $q \in Q$.

We say that a state is *absorbing* if its only successor in the transition graph is itself. A parity automaton has a *safety* acceptance condition if all of its states have priority 0, except for the additional absorbing `reject` state that has priority 1. An automaton with a safety acceptance condition is stratified, and hence safety conditions are a special case of weak conditions.

Whether an infinite word $w = w_0 w_1 w_2 \cdots \in \Sigma^\omega$ is *accepted* or *rejected* by an alternating automaton $\mathcal{A}$ is determined by the winner of the following *acceptance game* $\mathcal{G}(\mathcal{A}, w)$. The set of positions in the game is the set $Q \times \mathbb{N}$ and the two players, Alice and Elvis, play in the following way. The *initial position* is $(q_0, 0)$; for every *current position* (q_i, i), first Elvis chooses a subset P of Q that satisfies $\delta(q_i, w_i)$, then Alice picks a state $q_{i+1} \in P$, and $(q_{i+1}, i+1)$ becomes the next current position. Note that Elvis can be thought of making the non-deterministic choices and Alice can be thought of making the universal choices in the transition function of the alternating automaton. This interaction of Alice and Elvis yields an infinite sequence of states $q_0, q_1, q_2, \dots$, and whether Elvis is declared the winner or not is determined by whether the sequence is *accepting* according to the acceptance condition of the automaton. Acceptance games for parity, Büchi, co-Büchi, and weak conditions are parity games, which are *determined* [8]: in every acceptance game, either Alice or Elvis has a winning strategy. We say that an infinite word $w \in \Sigma^\omega$ is *accepted* by an alternating automaton $\mathcal{A}$ if Elvis has a winning strategy in the acceptance game $\mathcal{G}(\mathcal{A}, w)$, and otherwise it is *rejected.*

A *run dag* of an alternating automaton $\mathcal{A}$ on an infinite word w is a directed acyclic graph $G = (V, E, \rho : V \to Q)$, where $V \subseteq Q \times \mathbb{N}$ is the set of vertices; successors (according to the directed edge relation E) of every vertex (q, i) are of the form $(q', i+1)$; the following conditions hold:

- $(q_0, 0) \in V$,
- for every $(q, i) \in V$, the Boolean formula $\delta(q, w_i)$ is satisfied by the set of states p, such that $(p, i+1)$ is a successor of (q, i);

and ρ projects vertices onto the first component. Note that every vertex in a run dag has a successor, and hence every maximal path is infinite. We say that a run dag of an automaton $\mathcal{A}$ is *accepting* if the sequence of states on every infinite path in the run dag is accepting according to the accepting condition of $\mathcal{A}$. The *positional* determinacy theorem for parity games [8] implies that an infinite word w is accepted by an alternating automaton $\mathcal{A}$ with a parity (or Büchi, co-Büchi, weak, or safety) condition if and only if there is an accepting run dag of $\mathcal{A}$ on w. In other words, run dags are compact representations of (positional) winning strategies for Elvis in the acceptance games.

Run dags considered here are a special case of *layered dags*, whose vertices can be partitioned into sets $L_0, L_1, L_2, \ldots$, such that every edge goes from some layer L_i to the next layer L_{i+1}. We define the *width* of a layered dag with an infinite number of layers $L_0, L_1, L_2, \ldots$ to be $\liminf_{i\to\infty} |L_i|$. Note that the width of a run dag of an alternating automaton is trivially upper-bounded by the number of states of the automaton.

3 From co-Büchi and Büchi to weak

In this section we summarize the results of Kupferman and Vardi [15] who have given translations from alternating co-Büchi and Büchi automata to alternating weak automata with only a quadratic blow-up in the state space. We recall the decomposition of co-Büchi accepting run dags of Kupferman and Vardi in detail because it motivates and prepares the reader for our generalization of their result to accepting parity run dags. Our main technical result is a translation from alternating parity automata to alternating Büchi automata with only a quasi-polynomial blow-up in the state space, but the ultimate goal is a quasi-polynomial translation from parity to weak automata. Therefore, we also recall how Kupferman and Vardi use their quadratic co-Büchi to weak translation in order to obtain a quadratic Büchi to weak translation.

3.1 Co-Büchi progress measures

The main technical concept that underlies Kupferman and Vardi's [15] translation from alternating co-Büchi automata to alternating weak automata is that of a *ranking function* for accepting run dags of alternating co-Büchi automata. As Kupferman and Vardi themselves point out, ranking functions can be seen as equivalent to Klarlund's *progress measures* [12]. We will adopt Klarlund's terminology because the theory of progress measures for certifying parity conditions is very well developed [8, 12, 13, 10, 11, 5] and our main goal in this paper is to use a version of parity progress measures to give a simplified, streamlined, and improved translation from alternating parity to alternating weak automata.

Let $G = (V, E, \pi : V \to \{0, 1\})$ be a layered dag with vertex priorities 0 or 1, and in which every vertex has a successor. Note that all run dags of an alternating co-Büchi automaton are such layered dags and if the automaton has n states then the width of the run dag is at most n. (Observe, however, that while formally the third component $\rho : V \to Q$ in a run dag maps vertices to states, here we instead consider the labeling $\pi : V \to \{0, 1\}$ that labels vertices by the priorities of the states $\pi(v) = \pi(\rho(v))$.)

A *co-Büchi progress measure* [8, 13, 10] is a mapping $\mu : V \to M$, where $(M, \leq)$ is a well-ordered set, such that for every edge $(v, u) \in E$, we have

1. if $\pi(v) = 0$ then $\mu(v) \geq \mu(u)$,
2. if $\pi(v) = 1$ then $\mu(v) > \mu(u)$.

It is elementary to argue that existence of a co-Büchi progress measure on a graph is sufficient for every infinite path in the graph satisfying the co-Büchi condition. Importantly, it is also necessary, which can be, for example, deduced from the proof of positional determinacy for parity games due to Emerson and Jutla [8]. In other words, co-Büchi progress measures are witnesses for the property that all infinite paths in a graph satisfy the co-Büchi condition. The appeal of such witnesses stems from the property that while certifying a global and infinitary condition, it suffices to verify them locally by checking a simple inequality between the labels of the source and the target of each edge in the graph.

The disadvantage of progress measures as above is that on graphs of infinite size, such as run dags, the well-ordered sets of labels that are needed to certify co-Büchi conditions may be of unbounded (and possibly infinite) size. In order to overcome this disadvantage, and to enable automata-theoretic uses of progress measure certificates, Klarlund has proposed the following concept of lazy progress measures [12]. A *lazy (co-Büchi) progress measure* is a mapping $\mu : V \to M$, where $(M, \leq)$ is a well-ordered set and $L \subset M$ is the set of *lazy-progress* elements, and such that:

1. for every edge $(v, u) \in E$, we have $\mu(v) \geq \mu(u)$;
2. if $\pi(v) = 1$ then $\mu(v) \in L$;
3. on every infinite path in G, there are infinitely many vertices v such that $\mu(v) \notin L$.

It is elementary to prove the following proposition.

▶ **Proposition 1.** *If a graph has a lazy co-Büchi progress measure then all infinite paths in it satisfy the co-Büchi condition.*

The following converse establishes the attractiveness of lazy co-Büchi progress measures for certifying the co-Büchi conditions on layered dags of bounded width, and hence for certifying accepting run dags of alternating co-Büchi automata.

▶ **Lemma 2** (Klarlund [12]). *If all infinite paths in a layered dag $(V, E, \pi : V \to \{0, 1\})$ satisfy the co-Büchi condition and the width of the dag is at most n, then there is a lazy co-Büchi progress measure $\mu : V \to M$, where $M = \{1, 2, \ldots, 2n\}$ and $L = \{2, 4, 6, \ldots, 2n\}$.*

Proof. We summarize a proof given by Kupferman and Vardi [15] that provides an explicit decomposition of the accepting run dag into (at most) $2n$ parts from which a lazy co-Büchi progress measure can be straightforwadly defined. The proof by Klarlund [12] is more succinct, but the former is more constructive and hence more transparent.

Observe that if all infinite paths satisfy the co-Büchi condition then there must be a vertex v whose all descendants (i.e., vertices to which there is a – possibly empty – path from v) have priority 0; call such vertices 1*-safe* in $G_1 = G$. Indeed, otherwise it would be easy to construct an infinite path with infinitely many occurrences of vertices of priority 1.

Let S_1 be the set of all the 1-safe vertices in G_1, and let $G'_1 = G_1 \setminus S_1$ be the layered dag obtained from G_1 by removing all vertices in S_1. Note that there is an infinite path in the subgraph of G_1 induced by S_1, and hence the width of G'_1 is strictly smaller than the width of G_1.

Let R_1 be the set of all vertices in G'_1 that have only finitely many descendants; call such vertices *transient* in G'_1. Let G_2 be the the layered dag obtained from G'_1 by removing all vertices in R_1. Since G_2 is a subgraph of G'_1, the width of G_2 is strictly smaller than the width of G_1. Moreover, G_2 shares the key properties with G_1: every vertex has a successor and hence all the maximal paths are infinite, and all infinite paths satisfy the co-Büchi condition.

By applying the same procedure to G_2 that we have described for G_1 above, we obtain the set S_2 of 1-safe vertices in G_2 and the set R_2 of vertices transient in G'_2, and the layered dag G_3 – obtained from G_2 by removing all vertices in $S_2 \cup R_2$ – has the width that is strictly smaller than that of G_2. We can continue in this fashion until the graph G_{k+1}, for some $k \geq 1$, is empty. Since the width of G is at most n, and the widths of graphs $G_1, G_2, \ldots, G_{k+1}$ are strictly decreasing, it follows that $k \leq n$.

We define $\mu : V \to \{ 1, 2, \ldots, 2n \}$ by:

$$\mu(v) = \begin{cases} 2i - 1 & \text{if } v \in S_i, \\ 2i & \text{if } v \in R_i, \end{cases}$$

and note that it is routine to verify that if we let $L = \{ 2, 4, \ldots, 2n \}$ be the set of lazy-progress elements then μ is a lazy co-Büchi progress measure. ◀

3.2 From co-Büchi and Büchi to weak

In this section we present a proof of the following result.

▶ **Theorem 3** (Kupferman and Vardi [15]). *There is a translation that given an alternating co-Büchi automaton with n states yields an equivalent alternating weak automaton with $O(n^2)$ states.*

Proof. It suffices to argue that, given an alternating co-Büchi automaton $\mathcal{A} = (Q, q_0, \Sigma, \delta, \pi : Q \to \{ 0, 1 \})$ with n states, we can design an alternating weak automaton with $O(n^2)$ states that guesses and certifies a dag run of $\mathcal{A}$ together with a lazy co-Büchi progress measure on it as described in Lemma 2. First we construct a *safety* automaton $\mathcal{S}$ with $O(n^2)$ states that simulates the automaton $\mathcal{A}$ while guessing a lazy co-Büchi progress measure and verifying conditions 1) and 2) of its definition. Condition 3) will be later handled by turning the safety automaton $\mathcal{S}$ into a weak automaton $\mathcal{W}$ by appropriately assigning odd or even priorities to all states in $\mathcal{S}$. We split the design of $\mathcal{W}$ into those two steps so that we can better motivate and explain the generalized constructions in Section 5.

The safety automaton $\mathcal{S}$ has the following set of states:

$$Q \times \{ 2, 4, \ldots, 2n \} \qquad \cup \left(\pi^{-1}(0) \times \{ 1, 3, \ldots, 2n-1 \}\right) \cup \{ \texttt{reject} \} ;$$

its initial state is $(q_0, 2n)$; and its transition function δ' is obtained from the transition function δ of $\mathcal{A}$ in the following way: for every state (q, i), and for every $a \in \Sigma$, the formula $\delta'\big((q, i), a\big)$ is obtained from $\delta(q, a)$ by replacing every occurrence of state $q' \in Q$ by the disjunction (i.e., a non-deterministic choice)

$$(q', i) \lor (q', i-1) \lor \cdots \lor (q', 1) \tag{1}$$

where every occurrence (q', j) for which $\pi(q') = 1$ and j is odd stands for the state `reject`.

In other words, the safety automaton $\mathcal{S}$ can be thought of as consisting of $2n$ copies $\mathcal{A}_{2n}, \mathcal{A}_{2n-1}, \ldots, \mathcal{A}_1$ of $\mathcal{A}$, with the non-accepting states $\pi^{-1}(1)$ removed from the odd-indexed copies $\mathcal{A}_{2n-1}, \mathcal{A}_{2n-3}, \ldots, \mathcal{A}_1$, and in whose acceptance games, Elvis always has the choice to stay in the current copy of $\mathcal{A}$ or to move to one of the lower-indexed copies of $\mathcal{A}$. Since the transitions of the safety automaton $\mathcal{S}$ always respect the transitions of the original co-Büchi automaton $\mathcal{A}$, an accepting run dag of $\mathcal{S}$ yields a run dag of $\mathcal{A}$ (obtained from the first components of the states (q, i)) and a labelling of its vertices by numbers in $\{ 1, 2, \ldots, 2n \}$ (obtained from the second components of the states (q, i)). It is routine to verify that the design of the state set and of the transition function of the safety automaton $\mathcal{S}$ guarantees that the latter labelling satisfies conditions 1) and 2) of the definition of a lazy co-Büchi progress measure, where the set of lazy-progress elements is $\{ 2, 4, \ldots, 2n \}$.

By setting the state priority function $\pi' : (q, i) \mapsto i + 1$ for all non-`reject` states in $\mathcal{S}$, and $\pi' : \texttt{reject} \mapsto 1$, we obtain from $\mathcal{S}$ an automaton $\mathcal{W}$ whose acceptance condition is weak because – by design – the transition function is non-increasing w.r.t. the state priority function. One can easily verify that the addition of this weak acceptance condition to $\mathcal{S}$ allows the resulting automaton $\mathcal{W}$, for every input word, to guess and verify a lazy progress measure – if one exists – on a run dag of automaton $\mathcal{A}$ on the input word, while $\mathcal{W}$ rejects the input word otherwise. This completes our summary of the proof of Theorem 3. ◀

▶ **Corollary 4** (Kupferman and Vardi [15]). *There is a translation that given an alternating Büchi automaton with n states yields an equivalent alternating weak automaton with $O(n^2)$ states.*

The argument of Kupferman and Vardi is simple and it exploits the ease with which alternating automata can be complemented. Given an alternating Büchi automaton $\mathcal{A}$ with n states, first complement it with no state space blow-up, obtaining an alternating co-Büchi automaton with n states, next use the translation from Theorem 3 to obtain an equivalent alternating weak automaton with $O(n^2)$ states, and finally complement the latter again with no state space blow-up, hence obtaining an alternating weak automaton that is equivalent to $\mathcal{A}$ and that has $O(n^2)$ states.

4 Lazy parity progress measures

Before we introduce *lazy parity progress measures*, we recall the definition of (standard) parity progress measures [11, 5]. We define a *well-ordered tree* to be a finite prefix-closed set of sequences of elements of a well-ordered set. We call such sequences *nodes* of the tree, and their components are *branching directions.* We use the standard ancestor-descendant terminology to describe relative positions of nodes in a tree. For example, $\langle\rangle$ is the *root*; node $\langle x, y\rangle$ is the *child* of the node $\langle x\rangle$ that is reached from it via the branching direction y; node $\langle x, y\rangle$ is the *parent* of node $\langle x, y, z\rangle$; nodes $\langle x, y\rangle$ and $\langle x, y, z\rangle$ are *descendants* of nodes $\langle\rangle$ and $\langle x\rangle$; nodes $\langle\rangle$ and $\langle x\rangle$ are *ancestors* of nodes $\langle x, y\rangle$ and $\langle x, y, z\rangle$; and a node is a *leaf* if it does not have any children. All nodes in a well-ordered tree are well-ordered by the *lexicographic order* that is induced by the well-order on the branching directions; for example, we have $\langle x\rangle < \langle x, y\rangle$, and $\langle x, y, z\rangle < \langle x, w\rangle$ if $y < w$. We define the *depth* of a node to be the number of elements in the eponymous sequence, the *height* of a tree to be the maximum depth of a node, and the *size* of a tree to be the number of its nodes.

Parity progress measures assign labels to vertices of graphs with vertex priorities, and the labels are nodes in a well-ordered tree. A *tree labelling* of a graph with vertex priorities that do not exceed a positive even integer d is a mapping from vertices of the graph to nodes in a well-ordered tree of height at most $d/2$. We write $\langle m_{d-1}, m_{d-3}, \ldots, m_\ell\rangle$, for some odd ℓ, $1 \le \ell < d$, to denote such nodes. We say that such a node has an (odd) *level* ℓ and an (even) level $\ell - 1$, and the root $\langle\rangle$ has level d. Moreover, for every priority p, $0 \le p \le d$, we define the *p-truncation* $\langle m_{d-1}, m_{d-3}, \ldots, m_\ell\rangle|_p$ in the following way:

$$\langle m_{d-1}, m_{d-3}, \ldots, m_\ell\rangle|_p = \begin{cases} \langle m_{d-1}, m_{d-3}, \ldots, m_\ell\rangle & \text{for } p \le \ell, \\ \langle m_{d-1}, m_{d-3}, \ldots, m_{p+1}\rangle & \text{for even } p > \ell, \\ \langle m_{d-1}, m_{d-3}, \ldots, m_p\rangle & \text{for odd } p > \ell. \end{cases}$$

We then say that a tree labelling μ of a graph $G = (V, E)$ with vertex priorities $\pi : V \to \{0, 1, 2, \ldots, d\}$ is a *parity progress measure* if the following *progress condition* holds for every edge $(v, u) \in E$:

1. if $\pi(v)$ is even then $\mu(v)|_{\pi(v)} \ge \mu(u)|_{\pi(v)}$;
2. if $\pi(v)$ is odd then $\mu(v)|_{\pi(v)} > \mu(u)|_{\pi(v)}$.

It is well-known that satisfaction of such local conditions on every edge in a graph is sufficient for every infinite path in the graph satisfying the parity condition [10, 11]. Less obviously, it is also necessary, which can be, again, deduced from the proof of positional determinacy of parity games due to Emerson and Jutla [8]. In other words, parity progress measures are witnesses for the property that all infinite paths in a graph satisfy the parity condition. Like for the simpler co-Büchi condition, their appeal stems from the property that they certify conditions that are global and infinitary by verifying conditions that are local to every edge in the graph.

Similar to the simpler co-Büchi progress measures, parity progress measures may unfortunately require unbounded or even infinite well-ordered trees to certify parity conditions on infinite graphs, and hence we consider lazy parity progress measures, also inspired by Klarlund's pioneering work [12]. A *lazy tree* is a well-ordered tree with a distinguished subset of its nodes called *lazy nodes.* For convenience, we assume that only leaves may be lazy and the root never is.

A *lazy parity progress measure* is a tree labelling μ of a graph (V, E), where the labels are nodes in a lazy tree T, such that:

1. for every edge $(v, u) \in E$, $\mu(v)|_{\pi(v)} \geq \mu(u)|_{\pi(v)}$;
2. if $\pi(v)$ is odd then node $\mu(v)$ is lazy and its level is at least $\pi(v)$;
3. on every infinite path in G, there are infinitely many vertices v, such that $\mu(v)$ is not lazy.

First we establish that existence of a lazy progress measure is sufficient for all infinite paths in a graph to satisfy the parity condition.

▶ **Lemma 5.** *If a graph has a lazy parity progress measure then all infinite paths in it satisfy the parity condition.*

Proof. For the sake of contradiction, assume that there is an infinite path $v_1, v_2, v_3, \ldots$ in the graph for which the highest priority p that occurs infinitely often is odd. Let $i \geq 1$ be such that $\pi(v_j) \leq p$ for all $j \geq i$. By condition 1), we have:

$$\mu(v_i)|_p \geq \mu(v_{i+1})|_p \geq \mu(v_{i+2})|_p \geq \ldots . \tag{2}$$

Let $i \leq i_1 < i_2 < i_3 < \cdots$ be such that $\pi(v_{i_k}) = p$ for all $k = 1, 2, 3, \ldots$. By condition 2), all labels $\mu(v_{i_k})$, for $k = 1, 2, 3, \ldots$, are lazy and their level in the tree is at least p. By condition 3), for infinitely many k, $\pi(v_k)$ is not lazy, so infinitely many of the inequalities in (2) must be strict, which contradicts the well-ordering of the tree T. ◀

Now we argue that existence of lazy parity progress measure is also necessary for a graph to satisfy the parity condition. Moreover, we explicitly quantify the size of a lazy ordered tree the labels from which are sufficient to give a lazy progress measure for a layered dag, as a function of the width of the dag. Before we do that, however, we introduce a simple operation that we call a lazification of a finite ordered tree. If T is a finite ordered tree, then its *lazification* lazi(T) is a finite lazy tree that is obtained from T in the following way:

- all nodes in T are also nodes in lazi(T) and they are not lazy;
- for every non-leaf node t in T, t has extra lazy children in the tree lazi(T), one smaller and one larger than all the other children, and one in-between every pair of consecutive children.

It is routine to argue that if a tree has n leaves and it is of height at most h then its lazification lazi(T) has $O(nh)$ nodes and it is also of height h.

▶ **Theorem 6** (Klarlund [12]). *If all infinite paths in a layered dag satisfy the parity condition and the width of the dag is at most n, then there is a lazy parity progress measure whose labels are nodes in a tree that is a lazification of an ordered tree with at most n leaves.*

Proof. Klarlund's proof [12] is very succinct and it heavily relies on the result of Klarlund and Kozen on Rabin measures [13]. Our proof is not only self-contained but it also is more constructive and transparent. The hierarchical decomposition describes the fundamental structure of accepting run dags of alternating parity automata and it may be of independent interest. The argument presented here is a generalization of the proof of Lemma 2 – given in Section 3 – from co-Büchi conditions to parity conditions.

Consider a layered dag $G = (V, E, \pi)$ where $\pi : V \to \{0, 1, 2, \ldots, d\}$. For a priority p, $0 \le p < d$, we write $G^{\le p}$ for the subgraph induced by the vertices whose priority is at most p.

We describe the following decomposition of G. Let D be the set that consists of all vertices of the top even priority d in G, and R_0 all those vertices in the subgraph $G^{\le d-1}$ that have finitely many descendants. We say that those vertices are $(d-1)$-*transient* in $G^{\le d-1}$. In other words, $D \cup R_0$ is the set of vertices from which every path reaches (possibly immediately) a vertex of priority d.

Let $G_1 = G \setminus (D \cup R_0)$ be the layered dag obtained from G by removing all vertices in $D \cup R_0$. Observe that every vertex in G_1 has at least one successor and hence – unless G_1 is empty – all maximal paths are infinite. W.l.o.g., assume henceforth that G_1 is not empty. We argue that there must be a vertex in G_1 whose all descendants have priorities at most $d-2$; call such vertices $(d-1)$-*safe* in G_1. Indeed, otherwise it would be easy to construct an infinite path with infinitely many occurrences of the odd priority $d-1$ and no occurrences of the top even priority d.

Let S_1 be the set of all the $(d-1)$-safe vertices in G_1. Let H_1 be the subgraph of G induced by S_1, let n_1 be the width of H_1, and note that $n_1 > 0$. Set $G_1' = G_1 \setminus S_1$ to be the layered dag obtained from G_1 by removing all $(d-1)$-safe vertices in G_1.

Let R_1 be the set of all vertices in G_1' that have only finitely many descendants; call such vertices $(d-1)$-transient in G_1'. Finally, let G_2 be the layered dag obtained from G_1' by removing all the $(d-1)$-transient vertices in G_1'. Note that the width of G_2 is smaller than the width of G_1 by at least $n_1 > 0$.

Unless graph G_2 is empty, we can now apply the same steps to G_2 that we have described for G_1, and obtain:

- the set S_2 of $(d-1)$-safe vertices in G_2;
- the subgraph H_2 of G_2 induced by S_2, which is of width $n_2 > 0$;
- the layered dag G_2', obtained from G_2 by removing all the vertices in S_2;
- the set R_2 of $(d-1)$-transient vertices in G_2';
- the layered graph G_3, obtained from G_2 by removing all vertices in $S_2 \cup R_2$, and whose width is smaller than the width of G_2 by at least $n_2 > 0$.

We can continue in this fashion, obtaining graphs $G_1, G_2, \ldots, G_{k+1}$, until the graph G_{k+1}, for some $k \ge 0$, is empty. Since the width of G is at most n and the widths of the graphs $G_1, G_2, \ldots, G_k$ are positive (unless $k = 0$), we have that $k \le n$ and $\sum_{i=1}^{k} n_i \le n$.

The proces described above yields a hierarchical decomposition of the layered dag; we now define – by induction on d – the tree that describes the shape of this decomposition. We then argue that the lazification of this tree provides the set of labels in a lazy parity progress measure.

In the base case $d = 0$, the shape of the decomposition is the well-ordered tree T of height $h = 0/2 = 0$ with only a root node $\langle\rangle$. It is straightforward to see that the function that maps every vertex onto the root is a (lazy) progress measure.

For $d \geq 2$, note that all vertices in dags $H_1, H_2, \ldots, H_k$ have priorities at most $d-2$. By the inductive hypothesis, there are trees $T_1, T_2, \ldots, T_k$, of heights at most $h-1 = (d-2)/2$ and with at most $n_1, n_2, \ldots, n_k$ leaves, respectively, which are the shapes of the hierarchical decompositions of dags $H_1, H_2, \ldots, H_k$, respectively.

We now construct the finite ordered tree T of height at most $h = d/2$ that is the shape of the hierarchical decomposition of G: let T consist of the root node $\langle\rangle$ that has k children, which are the roots of the subtrees $T_1, T_2, \ldots, T_k$, in that order. Note that the number of leaves of T is at most $\sum_{i=1}^{k} n_i \leq n$. Consider the following mapping from vertices in the graph onto nodes in the lazification $\mathrm{lazi}(T)$ of tree T:

- vertices in set D are mapped onto the root of $\mathrm{lazi}(T)$;
- vertices in transient sets $R_0, R_1, R_2, \ldots, R_k$ are mapped onto the lazy children of the root of $\mathrm{lazi}(T)$: those in R_0 onto the smallest lazy child, those in R_1 onto the lazy child between the roots of T_1 and T_2, etc.;
- vertices in subgraphs $H_1, H_2, \ldots, H_k$ are inductively mapped onto the appropriate nodes in the lazy subtrees of $\mathrm{lazi}(T)$ that are rooted in the k non-lazy children of the root.

It is easy to verify that this mapping satisfies conditions 1) and 2) of the definition of a lazy parity progress measure. Condition 3) is ensured by the fact that the root of T is not lazy and by the inductive hypothesis. Recall that every infinite path satisfies the parity condition, thus the highest priority p seen infinitely often on a given path is even. If $p = d$, the path visits infinitely often vertices labelled by the root of T. Otherwise, eventually the path contains only vertices in one of the sets S_i and we can use the inductive hypothesis. ◀

5 From parity to Büchi via universal trees

In this section we complete the proof of the main technical result of the paper, which is a quasi-polynomial translation from alternating parity automata to alternating weak automata. The main technical tools that we use to design our translation are lazy progress measures and universal trees [11, 5], and the state space blow-up of the translation is merely quadratic in the size of the smallest universal tree. Nearly tight quasi-polynomial upper and lower bounds have recently been given for the size of the smallest universal trees [11, 5] and, in particular, they imply that if the number of priorities in a family of alternating parity automata is at most logarithmic in the number of states, then the state space blow-up of our translation is only polynomial.

▶ **Theorem 7.** *There is a translation that given an alternating parity automaton with n states and d priorities yields an equivalent alternating weak automaton whose number of states is polynomial if $d = O(\log n)$ and it is $n^{O(\lg(d/\lg n))}$ if $d = \omega(\log n)$.*

Before we proceed to prove the theorem, we recall the notion of universal ordered trees. An (n, h)*-universal (ordered) tree* [5] is an ordered tree, such that every finite ordered tree of height at most h and with at most n leaves can be isomorphically embedded into it. In such an embedding, the root of the tree must be mapped onto the root of the universal tree, and the children of every node must be mapped – injectively and in an order-preserving way – onto the children of its image. In order to upper-bound the size of the blow-up in our parity to weak translation, we rely on the following upper bound on the size of the smallest universal trees.

▶ **Theorem 8** (Jurdziński and Lazić [11]). *For all positive integers n and h, there is an (n, h)-universal tree with at most quasi-polynomial number of leaves. More specifically, the number of leaves is polynomial in n if $h = O(\log n)$, and it is $n^{\lg(h/\lg n)+O(1)}$ if $h = \omega(\log n)$.*

We also note that Czerwiński et al. [5] have subsequently proved a nearly-matching quasi-polynomial lower bound, hence establishing that the smallest universal trees have quasi-polynomial size.

In order to prove Theorem 7, we establish the following lemma that provides a translation from alternating parity automata to alternating Büchi automata whose state-space blow-up is tightly linked to the size of universal trees.

▶ **Lemma 9.** *There is a translation that given an alternating parity automaton with n states and d priorities yields an equivalent alternating Büchi automaton whose number of states is $O(ndL_U)$ where L_U is the number of leaves in an $(n, d/2)$-universal ordered tree U.*

Note that Theorem 7 follows from Lemma 9 by Theorem 8 and Corollary 4.

Proof of Lemma 9. Given an alternating parity automaton $\mathcal{A} = (Q, q_0, \Sigma, \delta, \pi : Q \to \{0, 1, \ldots, d\})$ with n states, we now design an alternating Büchi automaton that guesses and certifies a dag run of $\mathcal{A}$ together with a lazy parity progress measure on it. As for the co-Büchi to weak case, we first construct a *safety* automaton that simulates the automaton $\mathcal{A}$ while guessing a lazy parity progress measure and verifying conditions 1) and 2) of its definition. Condition 3) will be later handled by turning the safety automaton into a Büchi automaton by appropriately assigning priorities 1 or 2 to all states in the safety automaton.

Below we give a general construction of an alternating Büchi automaton $\mathcal{B}_T$ from any lazy well-ordered tree T, and then we argue that the alternating parity automaton $\mathcal{A}$ is equivalent to the alternating Büchi automaton $\mathcal{B}_{\text{lazi}(U)}$, for every $(n, d/2)$-universal tree U.

Let T be a lazy tree of width n and height $d/2$. The construction is by induction on d. The safety automaton $\mathcal{S}_T$ has the following set of states, which are pairs of an element of Q and of a node in T.

- If $d = 0$, then the set of states of $\mathcal{S}_T$ is $(Q \times \{\langle\rangle\}) \cup \{\texttt{reject}\}$.
- Otherwise, let $\langle x_1\rangle, \langle x_2\rangle, \ldots, \langle x_k\rangle$ be the children of the root, and $1 \le i_1 < i_2 < \ldots < i_m \le k$ are the indices of its leaves that are lazy.

  For $i \notin \{i_1, i_2, \ldots, i_m\}$, let T_i be the lazy subtrees of T of height at most $d/2 - 1$ rooted in $\langle x_i\rangle$. By induction, for all i, we obtain an alternating Büchi automaton that is obtained from the lazy tree T_i and from the alternating parity automaton $\mathcal{A}$ restricted to the states of priority up to $d - 2$. Let Ω_i denote its set of non-`reject` states. They are pairs consisting of an element of Q and of a node in a tree of height $d/2 - 1$; the latter is a sequence $\langle m_{d-3}, m_{d-5}, \ldots, m_\ell\rangle$ of at most $d/2 - 1$ branching directions. Let Γ_i be the set consisting of the pairs $(q, \langle x_i, m_{d-3}, m_{d-5}, \ldots, m_\ell\rangle)$ for $(q, \langle m_{d-3}, m_{d-5}, \ldots, m_\ell\rangle) \in \Omega_i$. Set $Q^{(d)}$ (resp. $Q^{(<d)}$) the set of states of priority d (resp. $< d$) in $\mathcal{A}$. The states of $\mathcal{S}_T$ are defined as:

$$\Big(Q^{(d)} \times \{\langle\rangle\}\Big) \cup \Big(Q^{(<d)} \times \{\langle x_{i_1}\rangle, \langle x_{i_2}\rangle, \ldots, \langle x_{i_m}\rangle\}\Big) \cup \bigcup_{i=1}^{k} \Gamma_i \cup \{\texttt{reject}\}.$$

The initial state is (q_0, t) where t is the largest tuple such that (q_0, t) is a state. Let us now define the transition function: for every state (q, t), and for every $a \in \Sigma$, the formula $\delta'\big((q, t), a\big)$ is obtained from $\delta(q, a)$ by replacing every occurrence of state $q' \in Q$ by the disjunction (i.e., a non-deterministic choice)

$$\bigvee \{\, (q', t') \;:\; t|_{\pi(q)} \ge t'|_{\pi(q)} \,\} ,$$

where every occurrence (q', t') which is not in the set of states stands for the state `reject`.

In other words, the safety automaton $\mathcal{S}_T$ can be thought of as consisting of copies of $\mathcal{A}$, for each node of the tree T, in whose acceptance games Elvis always has the choice to stay in the current copy of $\mathcal{A}$ or to move to one of a smaller node with respect to the priority of the current state. Since the transitions of the safety automaton $\mathcal{S}_T$ always respect the transitions of the original parity automaton $\mathcal{A}$, an accepting run dag of $\mathcal{S}_T$ yields a run dag of $\mathcal{A}$ (obtained from the first components of the states (q, t)) and a labelling of its vertices by nodes in T (obtained from the second components of the states (q, t)). It is routine to verify that the design of the state set and of the transition function of the safety automaton $\mathcal{S}_T$ guarantees that the latter labelling satisfies condition 1) and 2) of the definition of a lazy parity progress measure.

In order to obtain the Büchi automaton $\mathcal{B}_T$ from the safety automaton $\mathcal{S}_T$, it suffices to appropriately assign priorities 1 and 2 to all states: we let the state `reject` and all states (q, t) such that t is a lazy node in tree T have priority 1, and we let all states (q, t) such that t is not a lazy node in tree T have priority 2. Note that this ensures that a run of $\mathcal{B}_T$ is accepting if and only if the tree labelling of a run dag of $\mathcal{A}$ that the underlying safety automaton $\mathcal{S}_T$ guesses – in the second component of its states – satisfies condition 3) of the definition of a lazy parity progress measure.

We now argue that if U is an $(n, d/2)$-universal tree then the alternating Büchi automaton $\mathcal{B}_{\text{lazi}(U)}$ is equivalent to the alternating parity automaton $\mathcal{A}$. Firstly, all words accepted by $\mathcal{B}_T$ for any finite lazy ordered tree T are also accepted by $\mathcal{A}$. This is because – as we have argued above – every accepting run dag of any such automaton $\mathcal{B}_T$ yields both a run dag of $\mathcal{A}$ (in the first state components) and a lazy parity progress measure on it (in the second state components), and the latter certifies that the former is accepting.

It remains to argue that every word accepted by $\mathcal{A}$ is also accepted by $\mathcal{B}_{\text{lazi}(U)}$. By Theorem 6, for every accepting run dag of $\mathcal{A}$, there is a lazy progress measure whose labels are nodes in a tree $\text{lazi}(T)$, where T is an ordered tree of height at most $d/2$ and with at most n leaves. It is routine to verify that if an ordered tree can be isomorphically embedded in another, then the same holds for their lazifications. By $(n, d/2)$-universality of U, it follows that $\text{lazi}(T)$ can be isomorphically embedded in $\text{lazi}(U)$. Therefore, for every word on which there is an accepting run dag of $\mathcal{A}$, the automaton $\mathcal{B}_{\text{lazi}(U)}$ has the capacity to guess the run dag of $\mathcal{A}$ and to guess and certify a lazy progress measure on it.

In order to conclude the $O(ndL_U)$ upper bound on the number of states of $\mathcal{B}_{\text{lazi}(U)}$, it suffices to observe that if the number of leaves in an ordered tree T of height h is L then the number of nodes in $\text{lazi}(T)$ is $O(hL)$. ◀

6 Open questions

Our use of universal trees to turn alternating parity automata into Büchi automata, like Boker and Lehtinen's [1] register technique, does not exploit alternations (although the further Büchi to weak translation does): all transitions that are not copied from the original automaton are non-deterministic. Can universal and non-deterministic choices be combined to further improve these translations? Can the long-standing $\Omega(n \log n)$ lower bound [20] be improved, for example by a combination of the full-automata technique of Yan [23] and the recent lower bound techniques for non-deterministic safety separating automata based on universal trees [5] and universal graphs [3]?

References

1 U. Boker and K. Lehtinen. On the way to alternating weak automata. In *FSTTCS 2018*, volume 122 of *LIPIcs*, pages 21:1–21:22. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018.

2 C. S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *STOC 2017*, pages 252–263. ACM, 2017.

3 T. Colcombet and N. Fijalkow. Universal graphs and good for games automata: New tools for infinite duration games. In *FoSSaCS 2019*, volume 11425 of *LNCS*, pages 1–26. Springer, 2019.

4 T. Colcombet, D. Kuperberg, C. Löding, and M. Vanden Boom. Deciding the weak definability of Büchi definable tree languages. In *CSL 2013*, volume 23 of *LIPIcs*, pages 215–230. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2013.

5 W. Czerwiński, L. Daviaud, N. Fijalkow, M. Jurdziński, R. Lazić, and P. Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. In *SODA 2019*, pages 2333–2349. ACM/SIAM, 2019.

6 L. Daviaud, M. Jurdziński, and R. Lazić. A pseudo-quasi polynomial algorithm for solving mean-payoff parity games. In *LICS 2018*, pages 325–334. IEEE, 2018.

7 D. Drusinsky and D. Harel. On the power of bounded concurrency I: Finite automata. *Journal of the ACM*, 41(3):517–539, 1994.

8 E. A. Emerson and C. Jutla. Tree automata, mu-calculus and determinacy. In *FOCS 1991*, pages 368–377. IEEE, 1991.

9 J. Fearnley, S. Jain, S. Schewe, F. Stephan, and D. Wojtczak. An ordered approach to solving parity games in quasi polynomial time and quasi linear space. In *SPIN 2017*, pages 112–121, 2017.

10 M. Jurdziński. Small progress measures for solving parity games. In *STACS 2000*, volume 1770 of *LNCS*, pages 290–301. Springer, 2000.

11 M. Jurdziński and R. Lazić. Succinct progress measures for solving parity games. In *LICS 2017*, page 9pp. IEEE, 2017.

12 N. Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *FOCS 1991*, pages 358–367. IEEE, 1991.

13 N. Klarlund and D. Kozen. Rabin measures. *Chicago J. Theor. Comput. Sci.*, pages 1–24, 1995. Article 3.

14 O. Kupferman and M. Y. Vardi. Weak alternating automata and tree automata emptiness. In *STOC 1998*, pages 224–233. ACM, 1998.

15 O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Log.*, 2(3):408–429, 2001.

16 O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

17 K. Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In *LICS 2018*, pages 639–648. IEEE, 2018.

18 K. Lehtinen and S. Quickert. Σ_2^μ is decidable for Π_2^μ. In *CiE 2017: Unveiling Dynamics and Complexity*, pages 292–303, 2017.

19 P. Lindsay. On alternating ω-automata. *Theoretical Computer Science*, 43:107–116, 1988.

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

21 D. E. Muller, A. Saoudi, and P. E. Schupp. Alternating automata, the weak monadic theory of the tree and its complexity. In *ICALP 1986*, volume 226 of *LNCS*, pages 275–283. Springer, 1986.

22 M. Skrzypczak and I. Walukiewicz. Deciding the topological complexity of Büchi languages. In *ICALP 2016*, volume 55 of *LIPIcs*, pages 99:1–99:13. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2016.

23 Q. Yan. Lower bounds for complementation of ω-automata via the full automata technique. *Logical Methods in Computer Science*, 4:1–20, 2008.

Good for Games Automata: From Nondeterminism to Alternation

Udi Boker
Interdisciplinary Center (IDC) Herzliya, Israel
udiboker@gmail.com

Karoliina Lehtinen
University of Liverpool, United Kingdom
https://cgi.csc.liv.ac.uk/~lehtinen/
k.lehtinen@liverpool.ac.uk

Abstract

A word automaton recognizing a language L is good for games (GFG) if its composition with any game with winning condition L preserves the game's winner. While all deterministic automata are GFG, some nondeterministic automata are not. There are various other properties that are used in the literature for defining that a nondeterministic automaton is GFG, including "history-deterministic", "compliant with some letter game", "good for trees", and "good for composition with other automata". The equivalence of these properties has not been formally shown.

We generalize all of these definitions to alternating automata and show their equivalence. We further show that alternating GFG automata are as expressive as deterministic automata with the same acceptance conditions and indices. We then show that alternating GFG automata over finite words, and weak automata over infinite words, are not more succinct than deterministic automata, and that determinizing Büchi and co-Büchi alternating GFG automata involves a $2^{\Theta(n)}$ state blow-up. We leave open the question of whether alternating GFG automata of stronger acceptance conditions allow for doubly-exponential succinctness compared to deterministic automata.

2012 ACM Subject Classification Theory of computation → Logic and verification

Keywords and phrases Good for games, history-determinism, alternation

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.19

Related Version A full version of the paper is available at https://arxiv.org/abs/1906.11624.

Funding *Udi Boker*: Israel Science Foundation grant 1373/16
Karoliina Lehtinen: EPSRC grant EP/P020909/1 (Solving Parity Games in Theory and Practice)

Acknowledgements We thank Orna Kupferman for suggesting to look into alternating good-for-games automata and for stimulating discussions on the subject, and Thomas Colcombet for introducing to us his work on alternating history-deterministic automata.

1 Introduction

In general, deterministic automata have better compositional properties than nondeterministic automata, making them better suited for applications such as synthesis. Unfortunately, determinization is complicated and involves an exponential state space blow-up. Nondeterministic automata that are *good for games* (GFG) have been heralded as a potential way to combine the compositionality of deterministic automata with the conciseness of nondeterministic ones. In this article we are interested in the question of whether the benefits of good-for-games automata extend to alternating automata, which, in general, can be double-exponentially more concise than deterministic automata.

The first hurdle of studying GFG alternating automata is to settle on definitions. Indeed, for nondeterminisic automata, several GFG-type properties have been invented independently under different names: good for games [17], good for trees [19], and history-determinism [11].

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 19; pp. 19:1–19:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

While Henzinger and Piterman introduced the idea of automata that compose well with games, in their technical development they preferred to use a *letter game* such that one player having a winning strategy in this game implies that the nondeterministic automaton composes well with games [17]. In a similar vein, Kupferman, Safra and Vardi considered already in 1996 a form of nondeterministic automata that resolves its nondeterminism according to the past by looking at tree automata for derived word languages [19]; this notion of *good for trees* was shown to be equivalent to the letter game [3]. Independently, Colcombet introduced history-determinism in the setting of nondeterministic cost automata [11], and later extended it to alternating automata [13]. He showed that history-determinism implies that the automaton is suitable for composition with other alternating automata, a seemingly stronger property than just compositionality with games. Although Colcombet further developed history-determinism for cost automata, here we only consider automata with ω-regular acceptance conditions.

As a result, in the literature there are at least five different definitions that characterize, imply, or are implied by a nondeterministic automaton composing well with games: composition with games, composition with automata, composition with trees, letter games and history-determinism. While some implications between them are proved, others are folklore, or missing. Furthermore, these definitions do not all generalize in the same way to alternating automata: compositionality with games and with automata are agnostic to whether the automaton is nondeterministic, universal or alternating, and hence generalize effortlessly to alternating automata; the letter-game and good-for-tree automata on the other hand generalize “naturally” in a way that treats nondeterminism and universality asymmetrically and hence need be adapted to handle alternation.

In the first part of this article, we give a coherent account of good-for-gameness for alternating automata: we generalize all the existing definitions from nondeterministic to alternating automata, and show them be equivalent. This implies that these are also equivalent for nondeterministic automata. While some of these equivalences were already folklore for nondeterministic automata, others are more surprising: compositionality with one-player games implies compositionality with two-player games and compositionality with automata, despite games being a special case of alternating automata and single-player games being a special case of games. We also show that in the nondeterministic case each definition can be relaxed to an asymmetric requirement: composition with universal automata or universal trees is already equivalent to composition with alternating automata and games.

In the second part of this article, we focus on questions of expressiveness and succinctness. The first examples of GFG automata were built on top of deterministic automata [17], and Colcombet conjectured that history-deterministic alternating automata with ω-regular acceptance conditions are not more concise than deterministic ones [12]. Yet, this has since been shown to be false: already GFG nondeterministic Büchi automata cannot be pruned into deterministic ones [3] and co-Büchi automata can be exponentially more concise than deterministic ones [18]. In general, nondeterministic GFG automata are in between nondeterministic and deterministic automata, having some properties from each [4].

Whether GFG alternating automata can be double exponentially more concise than deterministic automata is particularly interesting in the wake of quasi-polynomial algorithms for parity games. Indeed, since 2017 when Calude et al. brought down the upper bound for solving parity games from subexponential to quasi-polynomial [10], the automata-theoretical aspects of solving parity games with quasi-polynomial complexity have been studied in more depth [20, 6, 7, 16, 1, 15, 14]. In particular, Bojańczyk and Czerwiński [1], and Czerwiński et al. [15] describe the quasi-polynomial algorithms for solving parity games explicitly in terms of deterministic word automata that separate some word languages. A polynomial deterministic *or GFG* safety separating automaton for these languages would imply a polynomial algorithm for parity games. However, it is shown in [15] that the smallest possible such *nondeterministic* automaton is quasi-polynomial. As this lower bound only applies for nondeterministic automata, it is interesting to understand whether alternating GFG automata could be more concise.

As for expressiveness, we show that alternating GFG automata are as expressive as deterministic automata with the same acceptance conditions and indices. The proof extends the technique used in the nondeterministic setting [3], producing a deterministic automaton from the product of the automaton and the two transducers that model its history-determinism.

Regarding succinctness, we first show that GFG automata over finite words, as well as weak automata over infinite words, are not more succinct than deterministic automata. The proof builds on the property that minimal deterministic automata of these types have exactly one state for each Myhill-Nerode equivalence class, and an analysis that GFG automata of these types must also have at least one state for each such class.

We proceed to show that determinizing Büchi and co-Büchi alternating GFG automata involves a $2^{\theta(n)}$ state blow-up. The proof in this case is more involved, going through two main lemmas. The first shows that for alternating GFG Büchi automata, a history-deterministic strategy need not remember the entire history of the transition conditions, and can do with only remembering the prefix of the word read. The second lemma shows that the breakpoint (Miyano-Hayashi) construction, which is used to translate an alternating Büchi automaton into a nondeterministic one, preserves GFGness. We leave open the question of whether alternating GFG automata of stronger acceptance conditions allow for doubly-exponential succinctness compared to deterministic automata.

Due to lack of space, some proofs are omitted and can be found in the full version [8].

2 Preliminaries

Words and automata. An *alphabet* Σ is a finite nonempty set of letters, a finite (resp. infinite) *word* $u = u_0 \dots u_k \in \Sigma^*$ (resp. $w = w_0 w_1 \dots \in \Sigma^\omega$) is a finite (resp. infinite) sequence of letters from Σ. A *language* is a set of words, and the empty word is written ϵ.

An *alternating word automaton* is $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$, where Σ is a finite nonempty alphabet, Q is a finite nonempty set of states, $\iota \in Q$ is an initial state, $\delta : Q \times \Sigma \to \mathsf{B}^+(Q)$ is a transition function where $\mathsf{B}^+(Q)$ is the set of positive boolean formulas (*transition conditions*) over Q, and α is an acceptance condition, on which we elaborate below. For a state $q \in Q$, we denote by $\mathcal{A}^q$ the automaton that is derived from $\mathcal{A}$ by setting its initial state to q. $\mathcal{A}$ is nondeterministic (resp. universal) if all its transition conditions are disjunctions (resp. conjunctions), and it is deterministic if all its transition conditions are states.

There are various acceptance conditions, defined with respect to the set of states that (a path of) a run of $\mathcal{A}$ visits. Some of the acceptance conditions are defined on top of a labeling of $\mathcal{A}$'s states. In particular, the parity condition is a labeling $\alpha : Q \to \Gamma$, where Γ is a finite set of priorities and a path is accepting if and only if the highest priority seen infinitely often on it is even. A Büchi condition is the special case of the parity condition where $\Gamma = \{1, 2\}$; states of priority 2 are called *accepting* and of priority 1 *rejecting*, and then α can be viewed as the subset of accepting states of Q. Co-Büchi automata are dual, with $\Gamma = \{0, 1\}$. A weak automaton is a Büchi automaton in which every strongly connected component of the graph induced by the transition function consists of only accepting or only rejecting states.

In Sections 2–5, we handle automata with arbitrary ω-regular acceptance conditions, and thus consider α to be a mapping from Q to a finite set Γ, on top of which some further acceptance criterion is implicitly considered (as in the parity condition). In Section 6, we focus on weak, Büchi, and co-Büchi automata, and then view α as a subset of Q.

Games. A *finite* Σ*-arena* is a finite $\Sigma \times \{A, E\}$-labeled Kripke structure. An *infinite* Σ*-arena* is an infinite $\Sigma \times \{A, E\}$-labeled tree. Nodes with an A-label are said to belong to Adam; those with an E-label are said to belong to Eve. We represent a Σ-arena as $R = (V, X, V_E, C)$, where V is its set of nodes, $X \subseteq V \times V$ its transitions, $V_E \subseteq V$ the E-labeled nodes, $V \setminus V_E$ the A-labeled nodes and $C : V \to \Sigma$ its Σ-labeling function. We will assume that all states have a successor. An arena might be rooted at an initial position $v_\iota \in V$.

A play in R is an infinite path in R. A *game* is a Σ-arena together with a winning condition $W \subseteq \Sigma^\omega$. A play π is said to be winning for Eve in the game if the Σ-labels along π form a word in W. Else π is winning for Adam.

A *strategy* for Eve (Adam, resp.) is a function $\tau : V^* \to V$ that maps a history $v_0 \dots v_i$, i.e. a finite prefix of a play in R, to a successor of v_i whenever $v_i \in V_E$ ($v_i \notin V_E$). A play $v_0 v_1 \dots$ agrees with a strategy τ for Eve (Adam) if whenever $v_i \in V_E$ ($v_i \notin V_E$), we have $v_{i+1} = \tau(v_i)$. A strategy for Eve (Adam) is winning from a position $v \in V$ if all plays beginning in v that agree with it are winning for Eve (Adam). We say that a player wins the game from a position $v \in V$ if they have a winning strategy from v. If the game is rooted at v_ι, we say that a player wins the game if they win from v_ι.

All the games we consider have ω-regular winning conditions and are therefore determined and the winner has a finite-memory strategy [9]. Finite-memory strategies can be modeled by *transducers*. Given alphabets I and O, an (I/O)*-transducer* is a tuple $\mathcal{M} = (I, O, M, \iota, \rho, \chi)$, where M is a finite set of states (memories), $\iota \in M$ is an initial memory, $\rho : M \times I \to M$ is a deterministic transition function, and $\chi : M \to O$ is an output function. The strategy $\mathcal{M} : I^* \to O$ is obtained by following ρ and χ in the expected way: we first extend ρ to words in I^* by setting $\rho(\epsilon) = \iota$ and $\rho(u \cdot a) = \rho(\rho(u), a)$, and then define $\mathcal{M}(u) = \chi(\rho(u))$.

Products. An example, without labeling, of a synchronized product is given in Figure 1.

▶ **Definition 1** (Synchronized product). *The* synchronized product $R \times \mathcal{A}$ *between a* Σ*-arena* $R = (V, X, V_E, \mathcal{C})$ *and an alternating automaton* $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$ *with mapping* $\alpha : Q \to \Gamma$ *is a* $\Gamma \cup \{\bot\}$*-arena of which the states are* $V \times \mathsf{B}^+(Q)$ *and the successor relation is defined by:*

- (v, q)*, for a state* q *of* Q*, has successors* $(v', \delta(q, \mathcal{C}(v')))$ *for each successor* v' *of* v *in* R*;*
- $(v, b \wedge b')$ *and* $(v, b \vee b')$ *have two successors* (v, b) *and* (v, b')*;*
- *If* R *is rooted at* v *then the root of* $R \times \mathcal{A}$ *is* $(v, \delta(\iota, \mathcal{C}(v)))$*.*

The positions belonging to Eve are (v, b) *where either* b *is a disjunction, or* b *is a state in* Q *and* $v \in V_E$*. The label of* (v, b) *is* $\alpha(b)$ *if* b *is a state of* Q*, and* $\bot$ *otherwise.*

▶ **Definition 2** (Automata composition). *Given alternating automata* $\mathcal{B} = (\Sigma, Q^\mathcal{B}, \iota^\mathcal{B}, \delta^\mathcal{B}, \beta : Q^\mathcal{B} \to \Gamma)$ *and* $\mathcal{A} = (\Gamma, Q^\mathcal{A}, \iota^\mathcal{A}, \delta^\mathcal{A}, \alpha)$*, their* composition $\mathcal{B} \times \mathcal{A}$ *consists of the synchronized product automaton* $(\Sigma, Q^\mathcal{B} \times Q^\mathcal{A}, (\iota^\mathcal{B}, \iota^\mathcal{A}), \delta, \alpha')$*, where* $\alpha'(q^B, q^A) = \alpha(q^A)$ *and* $\delta((q^\mathcal{B}, q^\mathcal{A}), a)$ *consists of* $f(\delta^\mathcal{B}(q^B, a), q^\mathcal{A})$ *where:*

- $f(c \vee c', q) = f(c, q) \vee f(c', q)$
- $f(c \wedge c', q) = f(c, q) \wedge f(c', q)$
- $f(q', q) = g(q', \delta^\mathcal{A}(q, \beta(q'))$ *where*
- $g(q, c \vee c') = g(q, c) \vee g(q, c')$
- $g(q, c \wedge c') = g(q, c) \wedge g(q, c')$
- $g(q, q') = (q, q')$*.*

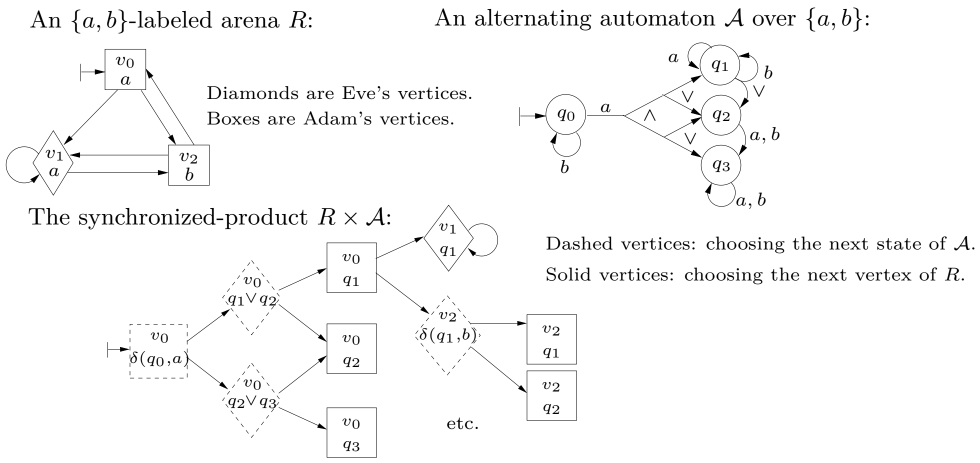

An $\{a,b\}$-labeled arena R:
An alternating automaton $\mathcal{A}$ over $\{a,b\}$:
Diamonds are Eve's vertices.
Boxes are Adam's vertices.
v_0 a
v_1 a
v_2 b
q_0 q_1 q_2 q_3
a b a,b a,b $\wedge$ $\vee$
The synchronized-product $R \times \mathcal{A}$:
Dashed vertices: choosing the next state of $\mathcal{A}$.
Solid vertices: choosing the next vertex of R.
v_0 $\delta(q_0,a)$
v_0 $q_1 \vee q_2$
v_0 $q_2 \vee q_3$
v_0 q_1
v_0 q_2
v_0 q_3
v_1 q_1
v_2 $\delta(q_1,b)$
v_2 q_1
v_2 q_2
etc.

Figure 1 An example of a product between an alternating automaton and a finite arena.

Note that this stands for first unfolding the transition condition in $\mathcal{B}$ and then the transition condition in $\mathcal{A}$, and it is equivalent to the following substitution, which matches Colcombet's notation [13]: $\delta^{\mathcal{B}}(q^{\mathcal{B}}, a)[q \in Q^{\mathcal{B}} \leftarrow \delta^{\mathcal{A}}(q^{\mathcal{A}}, \beta(q))[p \in Q^{\mathcal{A}} \leftarrow (q,p)]]$.

Acceptance of a word by an automaton. We define the acceptance directly in terms of the model-checking (membership) game, which happens to be exactly the product of the automaton with a path-like arena describing the input word. More precisely, $\mathcal{A}$ accepts a word w if and only if Eve wins the *model-checking game* $\mathcal{G}(w, \mathcal{A})$, defined as the product $R_w \times \mathcal{A}$, where the arena R_w consists of an infinite path, of which all positions belong to Eve (although it does not matter), and the label of the i^{th} position is the i^{th} letter of w. The language of an automaton $\mathcal{A}$, denoted by $L(\mathcal{A})$, is the set of words that it accepts (recognizes). Two automata are equivalent if they recognize the same language. We will refer to the positions of the arena R_w by the suffix of w that labels the path starting there. In particular, this gives us a finite representation of the arena for ultimately periodic words. We further denote by $\mathcal{G}_\tau(w, \mathcal{A})$ the model-checking game $\mathcal{G}(w, \mathcal{A})$ restricted to moves that agree with the strategy τ of Adam or Eve.

3 Good for Games Automata: Five Definitions

We clarify the definitions that are used in the literature for "good for gameness", while generalizing them from a nondeterministic to an alternating word automaton $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$.

Good for game composition. The first definition matches the intuition that "$\mathcal{A}$ is good for playing games". It was given in [17] for nondeterministic automata and applies as is to alternating automata, by properly defining the synchronized product of a game and an alternating automaton. (Definition 1.) We prove in Section 4 that it is equivalent when speaking of only one-player finite-arena games and two-player finite/infinite-arena games.

▶ **Definition 3** (Good for game composition). *$\mathcal{A}$ is* good for game composition *if for every [one-player] game G with a [finite] Σ-labeled arena and winning condition $L(\mathcal{A})$, Eve has a winning strategy in G if and only if she has a winning strategy in the synchronized-product game $G \times \mathcal{A}$.*

Compliant with the letter games. The first definition is simple to declare, but is not convenient for technical developments. Thus, Henzinger and Piterman defined the "letter-game", our next definition, while independently Colcombet defined history-determinism, which we provide afterwards. The two latter definitions are easily seen to be equivalent and they were shown in [17] to imply the game composition definition. We are not aware of a full proof of the other direction in the literature; we include one in this article.

In the letter-game for nondeterministic automata [17], Adam generates a word letter by letter, and Eve has to resolve the nondeterminism "on the fly", so that the generated run of $\mathcal{A}$ is accepting if Adam generates a word in the language. It has not been generalized yet to the alternating setting, and there are various ways in which it can be generalized, as it is not clear who should pick the letters and how to resolve the nondeterminism and universality. It turns out that a generalization that works well is to consider two independent games, one in which Eve resolves the nondeterminism while Adam picks the letters and resolves the universality, and another in which Eve picks the letters.

▶ **Definition 4** (Compliant with the letter games). *There are two letter games, Eve's game and Adam's game.*

Eve's game: *A configuration is a pair (σ, b) where $b \in \mathsf{B}^+(Q)$ is a transition condition and $\sigma \in \Sigma \cup \{\epsilon\}$ is a letter. (We abuse ϵ to also be an empty letter.) A play begins in $(\sigma_0, b_0) = (\epsilon, \iota)$ and consists of an infinite sequence of configurations $(\sigma_0, b_0)(\sigma_1, b_1)\ldots$. In a configuration (σ_i, b_i), the game proceeds to the next configuration (σ_{i+1}, b_{i+1}) as follows.*

- *If b_i is a state of Q, Adam picks a letter a from Σ, having $(\sigma_{i+1}, b_{i+1}) = (a, \delta(b_i, a))$.*
- *If b_i is a conjunction $b_i = b' \wedge b''$, Adam chooses between (ϵ, b') and (ϵ, b').*
- *If b_i is a disjunction $b_i = b' \vee b''$, Eve chooses between (ϵ, b') and (ϵ, b').*

In the limit, a play consists of an infinite sequence $\pi = b_0, b_1, \ldots$ of transition conditions and an infinite word w consisting of the concatenation of $\sigma_0, \sigma_1, \ldots$. Let ρ be the restriction of π to transition conditions that are states of Q. Eve wins the play if either $w \notin L(\mathcal{A})$ or ρ satisfies $\mathcal{A}$'s acceptance condition.

The nondeterminism in $\mathcal{A}$ is compliant with the letter games *if Eve wins this game.*

Adam's game *is similar to Eve's game, except that Eve chooses the letters instead of Adam, and Adam wins if either $w \in L(\mathcal{A})$ or ρ does not satisfy $\mathcal{A}$'s acceptance condition. The* universality in $\mathcal{A}$ is compliant with the letter games *if Adam wins this game.*

$\mathcal{A}$ is compliant with the letter games *if its nondeterminism and universality are compliant with the letter games.*

Observe that we need both games: universal automata are trivially compliant with Eve's letter game and nondeterministic automata are trivially compliant with Adam's letter game.

History-determinism. A nondeterministic automaton is history-deterministic [11] if there is a strategy[1] to resolve the nondeterminism that only depends on the word read so far, i.e., that is uniform for all possible futures. Colcombet generalized this to alternating automata [13] by considering such strategies for both players.

We first define how to use a strategy $\tau : (\Sigma \times \mathsf{B}^+(Q))^* \to \mathsf{B}^+(Q)$ for playing in a model-checking game $\mathcal{G}(w, \mathcal{A})$, as the history domains are different. Recall that in the model-checking game $\mathcal{G}(w, \mathcal{A})$, positions consist of a suffix of w and a transition condition

[1] In Section 2, we formally defined a "strategy" with respect to a specific game. Here we abuse the term "strategy" to refer to a general total function on finite words.

of $\mathcal{A}$ so histories have type $(\Sigma^\omega \times \mathsf{B}^+(Q))^*$. From such a history h, let h' be the history obtained by only keeping the first letter of the Σ^ω component of h's elements, that is, the letter at the head of the current suffix. Then, we extend τ to operate over the $(\Sigma^\omega \times \mathsf{B}^+(Q))^*$ domain, by defining $\tau(h) = \tau(h')$.

For convenience, we often refer to a history in $(\Sigma \times \mathsf{B}^+(Q))^*$, as a pair in $\Sigma^* \times (\mathsf{B}^+(Q))^*$.

▶ **Definition 5** (History-determinism [13]).

- *The* nondeterminism in $\mathcal{A}$ is history-deterministic *if there is a strategy* $\tau_E : (\Sigma\times\mathsf{B}^+(Q))^* \to \mathsf{B}^+(Q)$ *such that for all* $w \in L(\mathcal{A})$, τ_E *is a winning strategy for Eve in* $\mathcal{G}(w, \mathcal{A})$.
- *The* universality in $\mathcal{A}$ is history-deterministic *if there is a strategy* $\tau_A : (\Sigma \times \mathsf{B}^+(Q))^* \to \mathsf{B}^+(Q)$ *such that for all* $w \notin L(\mathcal{A})$, τ_A *is a winning strategy for Adam in* $\mathcal{G}(w, \mathcal{A})$.
- $\mathcal{A}$ *is* history-deterministic *if its nondeterminism and universality are history-deterministic.*

Good for automata composition. The next definition comes from Colcombet's proof that alternating history-deterministic automata behave well with respect to composition with other alternating automata. We shall show in Section 4 that it also implies proper compositionality with tree automata, and that for nondeterministic automata, it is enough to require compositionality with universal, rather than alternating, automata.

▶ **Definition 6** (Good for automata composition [13]). $\mathcal{A}$ *is* good for automata composition *if for every alternating word (or tree) automaton* $\mathcal{B}$ *with* Σ*-labeled states and acceptance condition* $L(\mathcal{A})$*, the language of the composed automaton* $\mathcal{B} \times \mathcal{A}$ *is equal to the language of* $\mathcal{B}$*.*

Good for trees. The next definition comes from the work in [19, 3] on the power of nondeterminism in tree automata. It states that a nondeterministic word automaton $\mathcal{A}$ is good-for-trees if we can "universally expand" it to run on trees and accept the "universally derived language" $L(\mathcal{A})\triangle$ – trees all of whose branches are in the word language of $\mathcal{A}$.

Observe that every universal word automaton for a language L is trivially good for $L\triangle$. Therefore, for universal automata, we suggest that a dual definition is more interesting: its "existential expansion to trees" accepts $L\triangledown$ – trees in which there exists a path in L.

For an alternating automaton $\mathcal{A}$, we generalize the good-for-trees notion to require that $\mathcal{A}$ is good for both $L(\mathcal{A})\triangle$ and $L(\mathcal{A})\triangledown$, when expanded universally and existentially, respectively.

We first formally generalize the definition of "expansion to trees" to alternating automata. The *universal (resp. existential) expansion* of $\mathcal{A}$ to trees is syntactically identical to $\mathcal{A}$, while its semantics is to accept a tree t if and only if Eve wins the game $t \times \mathcal{A}$, when t is viewed as a game in which all nodes belong to Adam (resp. Eve).

▶ **Definition 7** (Good for trees). $\mathcal{A}$ *is* good for trees *if its universal- and existential-expansions to trees recognize the tree languages* $L(A)\triangle$ *and* $L(A)\triangledown$*, respectively.*

4 Equivalence of All Definitions

We prove in this section the equivalence of all of the definitions in the alternating setting, as given in Section 3, which implies their equivalence also in the nondeterministic (and universal) setting. We may therefore refer to an automaton as *good-for-games (GFG)* if it satisfies any of these definitions. In some cases, we provide additional equivalences that only apply to the nondeterministic setting.

▶ **Theorem 8.** *Given an alternating automaton $\mathcal{A}$, the following are equivalent:*

1. *$\mathcal{A}$ is good for game composition (Def. 3).*
2. *$\mathcal{A}$ is compliant with the letter games (Def. 4).*
3. *$\mathcal{A}$ is history-deterministic (Def. 5).*
4. *$\mathcal{A}$ is good for automata composition (Def. 6).*
5. *$\mathcal{A}$ is good for trees (Def. 7).*

Proof.

- Lemma 9: History-determinism ⇔ compliance with the letter games. (Def. 4 ⇔ Def. 5).
- Lemma 13: Compliance with the letter games ⇒ compositionality with arbitrary games. (Def. 4 ⇒ "strong" Def. 3).
- Lemma 14: Compositionality, even with just one-player finite-arena games ⇒ compliance with the letter games. ("weak" Def. 3 ⇒ Def. 4).
- Lemma 15: Good for trees is ⇔ compositionality with one-player games. (Def. 7 ⇔ "medium" Def. 3).
- Lemma 16: Compositionality with games ⇔ compositionality with automata. (Def. 6 ⇔ Def. 3). ◀

We start with the simple equivalence of history-determinism and compliance with the letter game. (Observe that the letter-game strategies are of the same type as the strategies that witness history-determinism: a function from $(\Sigma \times \mathsf{B}^+(Q))^*$ to $\mathsf{B}^+(Q)$.)

▶ **Lemma 9.** *Consider an alternating automaton $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$.*

- *A strategy τ_E for Eve in her letter game is winning if and only if it witnesses the history-determinism of the nondeterminism in $\mathcal{A}$.*
- *A strategy τ_A for Adam in his letter game is winning if and only if it witnesses the history-determinism of the universality in $\mathcal{A}$.*
- *An alternating automaton $\mathcal{A}$ is history-deterministic if and only if it is compliant with the letter games.*

▶ **Corollary 10.** *If $\mathcal{A}$ is history-deterministic, then there are finite-memory strategies τ_E and τ_A to witness it.*

The following two propositions state that "standard manipulations" of alternating automata preserve history-determinism. The *dual* of an automaton $\mathcal{A}$, denoted by $\overline{\mathcal{A}}$, is derived from $\mathcal{A}$ by changing every conjunction of a transition condition to a disjunction, and vice versa, and changing the acceptance condition to reject every sequence of states (labeling) that $\mathcal{A}$ accepts, and accept every sequence that $\mathcal{A}$ rejects.

▶ **Proposition 11.** *Consider an alternating automaton $\mathcal{A}$ and its dual $\overline{\mathcal{A}}$. The nondeterminism (resp. universality) of $\mathcal{A}$ is history-deterministic if and only if the universality (resp. nondeterminism) of $\overline{\mathcal{A}}$ is history-deterministic.*

▶ **Proposition 12.** *Consider an alternating automaton $\mathcal{A}$, and let $\mathcal{A}'$ be an automaton that is derived from $\mathcal{A}$ by changing some transition conditions to different, but equivalent, boolean formulas. Then the nondeterminism/universality in $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$ is history-deterministic if and only if it is history-deterministic in $\mathcal{A}'$.*

The following lemma was shown in [17] for nondeterministic automata and can be deduced for alternating automata from Lemma 9 and the equivalence of history-determinism and being good for composition with automata [13]. We provide a simple direct proof.

▶ **Lemma 13.** *If an alternating automaton $\mathcal{A}$ is compliant with the letter games then it is good for game composition.*

If $\mathcal{A}$ is good for infinite games, it is clearly good for finite games, which can be unfolded into infinite games. The following lemma shows that the other direction holds too: compositionality with finite games implies compliance with the letter games, and therefore, from the previous lemma, composition with infinite games.

Note that this correspondence does not extend to the notion of *good for small games* [6, 14]: an automaton can be good for composition with games up to a bounded size, without being good for games.

▶ **Lemma 14.** *If an alternating automaton is good for composition with finite-arena one-player games then it is compliant with the letter games.*

Proof. Consider an alternating automaton $\mathcal{A}$ over the alphabet Σ. We show that if $\mathcal{A}$ is not compliant with the letter games then it is not good for finite-arena one-player games. By Proposition 12, we may assume that the transition conditions in $\mathcal{A}$ are in CNF.

If $\mathcal{A}$ is not compliant with Eve's letter game, then since this game is ω-regular, Adam has some finite-memory winning strategy, modeled by a transducer $\mathcal{M}$. Observe that states of $\mathcal{M}$ output the next moves of Adam, namely a letter and a disjunctive clause, and the transitions of $\mathcal{M}$ correspond to moves of Eve.

We translate $\mathcal{M}$ into a one-player Σ-labeled game with winning condition $L(\mathcal{A})$, in which all states belong to Adam: we consider every state/transition of $\mathcal{M}$ as a vertex/edge of $\mathcal{G}$, take the letter output of a state as the labeling of the corresponding vertex, and ignore the other outputs and the transition labels. (See an example in Figure 2.) We claim that $\mathcal{A}$ is not good for one-player finite-arena games, since i) Adam loses $\mathcal{G}$; and ii) Adam wins $\mathcal{G} \times \mathcal{A}$.

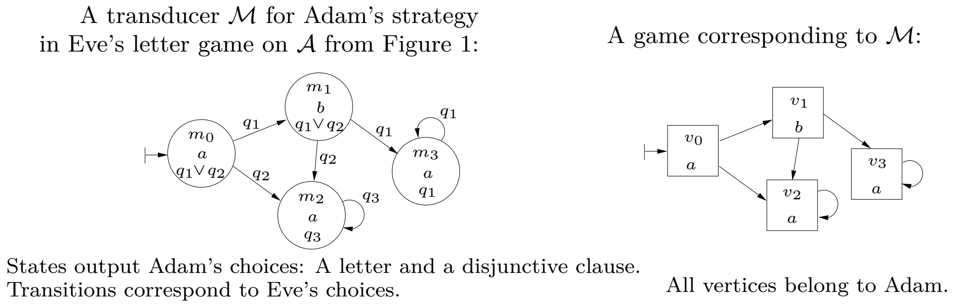
Indeed, considering claim (i), a play of $\mathcal{G}$ corresponds to a possible path in $\mathcal{M}$, which corresponds to a possible play in Eve's letter game that agrees with $\mathcal{M}$. If there is a play of $\mathcal{G}$ whose labeling is not in $L(\mathcal{A})$, it follows that Eve can win her letter game against $\mathcal{M}$, by forcing a word not in $L(\mathcal{A})$, which contradicts the assumption that $\mathcal{M}$ is a winning strategy.

As for claim (ii), Adam can play in the $\mathcal{G} \times \mathcal{A}$ game according to $\mathcal{M}$: whenever in a vertex (v, b) of $\mathcal{G} \times \mathcal{A}$, where v is a vertex of $\mathcal{G}$ and b a transition condition of $\mathcal{A}$, Adam chooses the next vertex according to the transition in the corresponding state in $\mathcal{M}$. Thus, the generated play in $\mathcal{G} \times \mathcal{A}$ corresponds to a play in Eve's letter game that agrees with $\mathcal{M}$, which Adam is guaranteed to win.

In the case that $\mathcal{A}$ is not compliant with Adam's letter game, we do the dual: Consider the transition conditions in $\mathcal{A}$ to be in DNF, have a winning strategy for Eve, modeled by a transducer $\mathcal{M}$ whose states output a letter and a conjunctive clause and whose transitions correspond to Adam's choices, and translate it to a Σ-labeled one-player game $\mathcal{G}$, in which all vertices belong to Eve. Then, for analogous reasons, Eve loses $\mathcal{G}$, but wins $\mathcal{G} \times \mathcal{A}$. ◀

The equivalence between the "good for trees" notion and being good for composition with one-player games, follows directly from the generalized definition of "good for trees" (Definition 7) and the following observation: Every one-player Σ-labeled game is built on top of a Σ-labeled tree (its arena, in case it is infinite, or the expansion of all possible plays, in case of a finite arena), and every Σ-labeled tree can be viewed as a one-player game by assigning ownership of all positions to either Adam or Eve. Clearly, every Σ-labeled tree t belongs to $L(A)\triangle$ if and only if Eve wins the game on t in which all nodes belong to Adam.

▶ **Lemma 15.** *An alternating automaton $\mathcal{A}$ is good for trees if and only if it is good for composition with one-player games.*


A transducer $\mathcal{M}$ for Adam's strategy in Eve's letter game on $\mathcal{A}$ from Figure 1:

A game corresponding to $\mathcal{M}$:

States output Adam's choices: A letter and a disjunctive clause. Transitions correspond to Eve's choices.

All vertices belong to Adam.

Figure 2 An example of a strategy for Adam in Eve's letter game, and the corresponding game, as used in the proof of Lemma 14.

A finite-arena game can be viewed as an alternating automaton over a singleton alphabet, suggesting that being good for composition with alternating automata implies being good for composition with finite games. This is indeed the case and by Lemmas 13 and 14, it also implies being good for infinite games. It turns out that even though alternating automata over a non-singleton alphabet cannot be just viewed as games, the other direction also holds.

▶ **Lemma 16.** *An alternating automaton $\mathcal{A}$ is good for game composition if and only if it is good for automata composition.*

Proof. We start with showing that being good for automata composition implies being good for game composition. Given a game over a finite Σ-arena $R = (V, X, V_E, L)$ with initial position ι and winning condition $W \subseteq \Sigma^\omega$, consider the automaton $A_R = (\{a\}, V, \iota, \delta, \alpha)$ over the alphabet $\{a\}$ with acceptance condition W, where $\delta(v, a) = \bigvee\{v' | (v, v') \in X\}$ if $v \in V_E$ and $\delta(v, a) = \bigwedge\{v' | (v, v') \in X\}$ otherwise. A_R accepts the unique word in $\{a\}^\omega$ if and only if Eve has a winning strategy in R from ι, because a strategy in R is exactly a run of A_R over this unique word, and it is winning if and only if the run is accepting.

Then, observing that the synchronized product $R \times \mathcal{A}$ between a finite game and an automaton is the special case of the synchronized product $A_R \times \mathcal{A}$, we conclude that if an automaton is good for automata composition, then Eve wins $R \times \mathcal{A}$ if and only if $A_R \times \mathcal{A}$ is non-empty, if and only if A_R is non-empty, i.e. if and only if Eve has a winning strategy in R. That is, $\mathcal{A}$ is good for finite game composition. From Lemmas 13 and 14, $\mathcal{A}$ is also good for composition with infinite games.

For the other direction, assume that $\mathcal{A}$ is good for game composition. We show that $\mathcal{A}$ is also good for automata composition. Consider an alternating automaton $\mathcal{B}$ with acceptance condition $L(\mathcal{A})$. Let $w \in L(\mathcal{B})$ and consider the model-checking game $\mathcal{G}(w, \mathcal{B})$ in which Eve has a winning strategy. Since $\mathcal{A}$ is good for game composition, Eve also has a winning strategy s in $\mathcal{G}(w, \mathcal{B}) \times \mathcal{A}$. We use this strategy to build a strategy s' for Eve in $\mathcal{G}(w, \mathcal{B} \times \mathcal{A})$.

First recall from Def. 2, that the transitions of $\mathcal{B} \times \mathcal{A}$ are of the form $f(c, q) \vee f(c', q)$, $f(c, q) \wedge f(c', q)$, corresponding to choices in $\mathcal{B}$, or of the form $g(q, c) \vee g(q, c')$ or $g(q, c) \wedge g(q, c')$, corresponding to choices in $\mathcal{A}$. At a position $(w, f(c, q) \vee f(c', q))$, Eve plays as s plays at $((w, c \vee c'), q)$; at $(w, g(q, c) \vee g(q, c'))$ Eve plays as s plays at $((w, c \vee c'), q)$. Since the winning condition in both games is determined by the states of $\mathcal{A}$ visited infinitely often, if s is winning, so is s'. Therefore $L(\mathcal{B} \times \mathcal{A})$ accepts w and $L(\mathcal{B}) \subseteq L(\mathcal{B} \times \mathcal{A})$. In the case $w \notin L(\mathcal{B})$, Adam can similarly copy his strategy from $\mathcal{G}(w, \mathcal{B}) \times \mathcal{A}$ into $\mathcal{G}(w, \mathcal{B} \times \mathcal{A})$.

We conclude that the language of $\mathcal{B} \times \mathcal{A}$ is equal to the language of $\mathcal{B}$ and therefore $\mathcal{A}$ is good for automata composition. ◀

▶ Remark 17. We observe that compositionality with word automata also implies compositionality with (symmetric, unranked) tree automata. A tree automaton is similar to a word automaton, except that its transitions have modalities $\Box q$ and $\Diamond q$ instead of states. Then, the model-checking (or membership) game of a tree and an automaton is a game, as for words, where, in addition, the modalities $\Box q$ and $\Diamond q$ dictate whether the choice of successor in the tree is given to Adam or Eve. Then, if $\mathcal{A}$ composes with games, it must in particular compose with the model-checking game of t and a tree automaton $\mathcal{B}$ with acceptance condition $L(\mathcal{A})$. If Eve (Adam) wins the model-checking game $\mathcal{G}(t, \mathcal{B})$, she (he) also wins $\mathcal{G}(t, \mathcal{B}) \times \mathcal{A}$. Her (his) winning strategy in this game is also a winning strategy in $\mathcal{G}(t, \mathcal{B} \times \mathcal{A})$, so $\mathcal{B} \times \mathcal{A}$ must accept (reject) t. $\mathcal{A}$ therefore composes with tree-automata.

While Theorem 8 obviously holds also for nondeterministic automata, we observe that in the absence of universality, the definitions of Section 3 can be relaxed into asymmetrical ones. For letter games, history-determinism, and good-for-trees, it follows directly from the definitions, as only their "nondeterministic part" applies. For composition with games and automata, we also show that it suffices to compose with universal automata and games.

▶ **Lemma 18.** *A nondeterministic automaton $\mathcal{A}$ is good for automata composition if and only if it is good for composition with universal automata.*

5 Expressiveness

For some acceptance conditions, such as weak, Büchi, and co-Büchi, alternating automata are more expressive than deterministic ones. For other conditions, such as parity, Rabin, Streett, and Muller, they are not. Yet, also for the latter conditions, once considering the condition's *index*, which is roughly its size, alternating automata are more expressive than deterministic automata with the same acceptance condition and index. (More details on the different acceptance conditions can be found, for example, in [2].)

Most acceptance conditions are preserved, together with their index, when taking the product of an automaton $\mathcal{A}$ with an auxiliary memory M. In such a product, the states of the resulting automaton are pairs (q, m) of a state q from $\mathcal{A}$ and a state m from M, while the acceptance condition is defined according to the projection of the states to their $\mathcal{A}$'s component. In particular, the weak, Büchi, co-Büchi, parity, Rabin, and Streett conditions are preserved, together with their index, under memory product, while the very-weak and Muller conditions are not.

For showing that GFG automata are not more expressive than deterministic automata with the same acceptance condition and index, we generalize the proof of [3] from nondeterminism to alternation. The idea is to translate an alternating GFG automaton $\mathcal{A}$ to an equivalent deterministic automaton $\mathcal{D}$ by taking the product of $\mathcal{A}$ with the transducers that model the history-deterministic strategies of $\mathcal{A}$.

▶ **Theorem 19.** *Every alternating GFG automaton with acceptance condition that is preserved under memory-product can be translated to a deterministic automaton with the same acceptance condition and index. In particular, this is the case for weak, co-Büchi, Büchi, parity, Rabin, and Streett GFG alternating automata of any index.*

Proof. Consider an alternating GFG automaton $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$. For convenience, we may assume by Proposition 12 that $\mathcal{A}$'s transition conditions are in DNF.

By Corollary 10, the history-determinism of $\mathcal{A}$'s universality and nondeterminism is witnessed by finite-memory strategies, modeled by transducers $M_A = (I_A, O_A, M_A, \iota_A, \rho_A, \chi_A)$ and $M_E = (I_E, O_E, M_E, \iota_E, \rho_E, \chi_E)$, respectively. Observe that since transition conditions of $\mathcal{A}$ are in DNF, the strategy $\mathcal{M}_A$ chooses a state of $\mathcal{A}$ for every letter in Σ and clause of states of $\mathcal{A}$, while the transitions in $\mathcal{M}_E$ are made in pairs, first choosing a clause for a letter in Σ and then updating the memory again according to Adam's choice of a state of $\mathcal{A}$.

Formally, we have that the elements of I_A are pairs (a, C), where $a \in \Sigma$ and C is a clause of states in Q, and that elements of O_A are states in Q, while elements of I_E are in $\Sigma \cup Q$ and elements of O_E are either clauses of states in Q or ϵ (when only updating the memory).

Let $\mathcal{D} = (\Sigma, Q', \iota', \delta', \alpha')$ be the deterministic automaton that is the product of $\mathcal{A}$ and $\mathcal{M}_A$, in which the universality is resolved according to $\mathcal{M}_A$ and the nondeterminism according to $\mathcal{M}_E$. That is, $Q' = Q \times M_A \times M_E$, $\iota' = (\iota, \iota_A, \iota_E)$, $\alpha'(q, x, y) = \alpha(q)$, and for every $q \in Q$, $x \in M_A$, $y \in M_E$, and $a \in \Sigma$, we have $\delta'((q, x, y), a) = (q', x', y')$, where $x' = \rho_A(x, (a, \chi_E(\rho_E(y, a))))$, $q' = \chi_A(x')$, and $y' = \rho_E(\rho_E(y, a), q')$.

Observe that $\mathcal{A}$ and $\mathcal{D}$ have the same acceptance condition and the same index, as $\mathcal{A}$'s acceptance condition is preserved under memory-product. We also have that $\mathcal{A}$ and $\mathcal{D}$ are equivalent, since for every word w, the games $\mathcal{G}(w, \mathcal{A})$, $\mathcal{G}_{\mathcal{M}_A, \mathcal{M}_E}(w, \mathcal{A})$, and $\mathcal{G}(w, \mathcal{D})$ have the same winner. ◀

6 Succinctness

Nondeterministic GFG automata over finite words and weak GFG automata over infinite words can be pruned to equivalent deterministic automata [19, 22]. We show that this remains true in the alternating setting. The succinctness of nondeterministic GFG Büchi automata compared to deterministic ones is still an open question, having no lower bound and a quadratic upper bound, whereas nondeterministic GFG co-Büchi automata can be exponentially more succinct than their deterministic counterparts [18]. We show that in the alternating setting, both Büchi and co-Büchi GFG automata are singly-exponential more succinct than deterministic ones. We leave open the question of whether stronger acceptance conditions can allow GFG automata to be doubly-exponential more succinct than deterministic ones.

In this section we focus on specific classes of automata, and for brevity use three letter acronyms in {D, N, A} × {W, B, C} × {W} when referring to them. The first letter stands for the transition mode (deterministic, nondeterministic, alternating); the second for the acceptance-condition (weak, Büchi, co-Büchi); and the third indicates that the automaton runs on words. For example, DBW stands for a deterministic Büchi automaton on words. We also use DFA, NFA, and WFA when referring to automata over finite words.

In the nondeterministic setting, the proof that GFG NFAs and GFG NWWs are not more succinct than DFAs and DWWs, respectively, is based on two properties: i) In a minimal DFA or DWW for a language L, there is exactly one state for every Myhill-Nerode equivalence class of L. (Recall that finite words u and v are in the same class C when for every word w, $uw \in L$ if and only if $vw \in L$. For a class C, the language $L(C)$ of C is $\{w \mid \exists u \in C \text{ such that } uw \in L\}$.); and ii) In a nondeterministic GFG automaton $\mathcal{A}$ that has no redundant transitions (removing a transition will change its language or make it not GFG), for every finite word u and states $q, q' \in \delta(u)$, we have $L(\mathcal{A}^q) = L(\mathcal{A}^{q'})$.

For showing that GFG AFAs and AWWs are not more succinct than DFAs and DWWs, respectively, we provide in the following lemma a variant of the above second property.

▶ **Lemma 20.** *Consider a GFG alternating automaton $\mathcal{A}$. Then for every class C of the Myhill-Nerode equivalence classes of $L(\mathcal{A})$, there is a state q in $\mathcal{A}$, such that $L(\mathcal{A}^q) = L(C)$.*

Proof. Let τ and η be history-deterministic strategies of $\mathcal{A}$ for Eve and Adam, respectively. For every finite word u, let $C(u)$ be the Myhill-Nerode equivalence class of u, and $q(u)$ be the state that $\mathcal{A}$ reaches when running on u along τ and η. We claim that $L(\mathcal{A}^{q(u)}) = L(C(u))$.

Indeed, if there is a word $w \in L(C(u)) \setminus L(\mathcal{A}^{q(u)})$ then Adam wins the model-checking game $\mathcal{G}_\tau(uw, \mathcal{A})$, by playing according to η until reaching $q(u)$ over u and then playing unrestrictedly for rejecting the w suffix, contradicting the history-determinism of τ.

Analogously, if there is a word $w \in L(\mathcal{A}^{q(u)}) \setminus L(C(u))$ then Eve wins the model-checking game $\mathcal{G}_\eta(uw, \mathcal{A})$, by playing according to τ until reaching $q(u)$ over u and then playing unrestrictedly for accepting the w suffix, contradicting the history-determinism of η. ◀

The insuccinctness of GFG AFAs and GFG AWWs directly follows.

▶ **Theorem 21.** *For every GFG AFA or GFG AWW $\mathcal{A}$, there is an equivalent DFA or DWW $\mathcal{A}'$, respectively, such that the number of states in $\mathcal{A}'$ is not more than in $\mathcal{A}$.*

As opposed to weak automata, minimal deterministic Büchi and co-Büchi automata do not have the Myhill-Nerode classification, and indeed, it was shown in [18] that GFG NCWs can be exponentially more succinct than DCWs.

We show that GFG ACWs are also only singly-exponential more succinct than DCWs. We translate a GFG ACW $\mathcal{A}$ to an equivalent DCW $\mathcal{D}$ in four steps: i) Dualize $\mathcal{A}$ to a GFG ABW $\mathcal{B}$; ii) Translate $\mathcal{B}$ to an equivalent NBW, having an $O(3^n)$ state blow-up [21, 5], and prove that the translation preserves GFGness; iii) Translate $\mathcal{B}$ to an equivalent DBW $\mathcal{C}$, having an additional quadratic state blow-up [18]; and iv) Dualize $\mathcal{C}$ to a DCW $\mathcal{D}$.

The main difficulty is, of course, in the second step, showing that the translation of an ABW to an NBW preserves GFGness. For proving it, we first need the following key lemma, stating that in a GFG ABW in which the transition conditions are given in DNF, the history-deterministic strategies can only use the current state and the prefix of the word read so far, ignoring the history of the transition conditions.

▶ **Lemma 22.** *Consider an ABW $\mathcal{A}$ with transition conditions in DNF and history-deterministic nondeterminism. Then Eve has a strategy $\tau : \Sigma^* \times Q \to \mathsf{B}^+(Q)$ (and not only a strategy $(\Sigma \times \mathsf{B}^+(Q))^* \to \mathsf{B}^+(Q)$), such that for every word $w \in L(\mathcal{A})$, Eve wins $\mathcal{G}_\tau(w, A)$.*

Proof. Let $\xi : (\Sigma \times \mathsf{B}^+(Q))^* \to \mathsf{B}^+(Q)$ be a "standard" history-deterministic strategy for Eve. Observe that since the transition conditions of $\mathcal{A}$ are in DNF, ξ's domain is $(\Sigma \times Q)^*$, and the run of $\mathcal{A}$ on w following ξ, namely $\mathcal{G}_\xi(w, \mathcal{A})$, is an infinite tree, in which every node is labeled with a state of $\mathcal{A}$. A history h for ξ is thus a finite sequence of states and a finite word. Let $\mathsf{yearn}(h)$ denote the number of positions in the sequence of states in h from the last visit to α until the end of the sequence. We shall say "a history h for a word u" when h's word component is u, and "h ends with q" when the sequence of states in h ends in q.

We inductively construct from ξ a strategy $\tau : \Sigma^* \times Q \to \mathsf{B}^+(Q)$, by choosing for every history $(u, q) \in \Sigma^* \times Q$ some history h of ξ, as detailed below, and setting $\tau(u, q) = \xi(h)$.

At start, for $u = \epsilon$, we set $\tau(\epsilon, \iota) = \xi(\epsilon, \iota)$. In a step in which every history of ξ for u ends with a different state, we set $\tau(u, q) = \xi(h)$, where h is the single history that ends with q.

The challenge is in a step in which several histories of ξ for u end with the same state, as τ can follow only one of them. We define that $\tau(u, q) = \xi(h)$, where h is a history of ξ for u that ends with q, and $\mathsf{yearn}(h)$ is maximal among the histories of ξ for u that ends in q. Every history of ξ that is not followed by τ is considered "stopped", and in the next iterations of constructing τ, histories of ξ with stopped prefixes will not be considered.

As ξ is a winning strategy for *Eve*, all paths in $\mathcal{G}_\xi(w, A)$ are accepting. Observe that $\mathcal{G}_\tau(w, A)$ is a tree in which some of the paths are from $\mathcal{G}_\xi(w, A)$ and some are not – whenever a history is stopped in the construction of τ, a new path is created, where its prefix is of the stopped history and the continuation follows the path it was redirected to. We will show that, nevertheless, all paths in $\mathcal{G}_\tau(w, A)$ are accepting.

Assume toward contradiction a path ρ of $\mathcal{G}_\tau(w, A)$ that is not accepting, and let k be its last position in α. The path ρ must have been created by infinitely often redirecting it to different histories of ξ, as otherwise there would have been a rejecting path of ξ. Now, whenever ρ was redirected, it was to a history h, such that $\mathsf{yearn}(h)$ was maximal. Thus, in particular, this history did not visit α since position k. Therefore, by König's lemma, there is a path π of $\mathcal{G}_\xi(w, A)$ that does not visit α after position k, contradicting the assumption that all paths of $\mathcal{G}_\xi(w, A)$ are accepting. ◀

We continue with showing that the translation of an ABW to an NBW preserves GFGness.

▶ **Lemma 23.** *Consider an ABW $\mathcal{A}$ for which the nondeterminism is history-deterministic. Then the nondeterminism in the NBW $\mathcal{A}'$ that is derived from $\mathcal{A}$ by the breakpoint (Miyano-Hayashi) construction is also history-deterministic.*

Proof. Consider an ABW $\mathcal{A} = (\Sigma, Q, \iota, \delta, \alpha)$ for which the nondeterminism is history-deterministic. We write $\overline{\alpha}$ for $Q \setminus \alpha$. By Proposition 12, we may assume that the transition conditions of $\mathcal{A}$ are given in DNF.

The breakpoint construction [21] generates from $\mathcal{A}$ an equivalent NBW $\mathcal{A}'$, by intuitively maintatining a pair $\langle S, O\rangle$ of sets of states of $\mathcal{A}$, where S is the set of states that are visited at the current step of a run, and O are the states among them that "owe" a visit to α. A state owes a visit to α if there is a path leading to it with no visit to α since the last "breakpoint", which is a state of $\mathcal{A}'$ in which $O = \emptyset$. The accepting states of $\mathcal{A}'$ are the breakpoints.

For providing the formal definition of $\mathcal{A}'$, we first construct from δ and each letter $a \in \Sigma$, a set $\Delta(a)$ of transition functions $\gamma_1, \gamma_2, \ldots \gamma_k$, for some $k \in \mathbb{N}$, such that each of them has only universality and corresponds to a possible resolution of the nondeterminism in every state of $\mathcal{A}$. For example, if $\mathcal{A}$ has states q_1, q_2, and q_3, and its transition function δ for the letter a is $\delta(q_1, a) = q_1 \wedge q_3 \vee q_2; \delta(q_2, a) = q_2 \vee q_3 \vee q_1$; and $\delta(q_3, a) = q_2 \wedge q_3 \vee q_1$, then $\Delta(a)$ is a set of twelve transition functions, where $\gamma_1(q_1, a) = q_1 \wedge q_3; \gamma_1(q_2, a) = q_2$; and $\gamma_1(q_3, a) = q_2 \wedge q_3$, etc., corresponding to the possible ways of resolving the nondeterminism in each of the states.

For convenience, we shall often consider a conjunctive formula over states as a set of states, for example $q_1 \wedge q_3$ as $\{q_1, q_3\}$. For a set of states $S \subseteq Q$, a letter a, and a transition function $\gamma \in \Delta(a)$, we define $\gamma(S) = \bigcup_{q \in S} \gamma(q, a)$.

Formally, the breakpoint construction [21] generates from $\mathcal{A}$ an equivalent NBW $\mathcal{A}' = (\Sigma, Q', \iota', \delta', \alpha')$ as follows:

- $Q' = \{\langle S, O\rangle \mid O \subseteq S \subseteq Q\}$
- $\iota' = (\{\iota\}, \{\iota\} \cap \overline{\alpha})$
- δ' : For a state $\langle S, O\rangle$ of $\mathcal{B}$ and a letter $a \in \Sigma$:
  - If $O = \emptyset$ then $\delta'(\langle S, O\rangle, a) = \{\langle \hat{S}, \hat{O}\rangle \mid$ exists a transition function $\gamma \in \Delta$, such that $\hat{S} = \gamma(S, a)$ and $\hat{O} = \gamma(S, a) \cap \overline{\alpha}\}$
  - If $O \neq \emptyset$ then $\delta'(\langle S, O\rangle, a) = \{\langle \hat{S}, \hat{O}\rangle \mid$ exists a transition function $\gamma \in \Delta$, such that $\hat{S} = \gamma(S, a)$ and $\hat{O} = \gamma(O, a) \cap \overline{\alpha}\}$
- $\alpha' = \{(S, \emptyset) \mid S \subseteq Q\}$

Observe that the breakpoint construction determinizes the universality of $\mathcal{A}$, while morally keeping its nondeterminism as is. This will allow us to show that Eve can use her history-deterministic strategy for $\mathcal{A}$ also for resolving the nondeterminism in $\mathcal{A}'$.

At this point we need Lemma 22, guaranteeing a strategy $\tau : \Sigma^* \times Q \to \mathsf{B}^+(Q)$ for Eve. At each step, Eve should choose the next state in $\mathcal{A}'$, according to the read prefix u and the current state (S, O) of $\mathcal{A}'$. Observe that τ assigns to every state $q \in S$ a set of states $S' = \tau(u, q)$, following a nondeterministic choice of $\delta(q, u)$; Together, all these choices comprise some transition function $\gamma \in \Delta$. Thus, in resolving the nondeterminism of $\mathcal{A}'$, Eve's strategy τ' is to choose the transition γ that is derived from τ.

Since τ guarantees that all the paths in the τ-run-tree of $\mathcal{A}$ on a word $w \in L(\mathcal{A})$ are accepting, the corresponding τ'-run of $\mathcal{A}'$ on w is accepting, as infinitely often all the $\mathcal{A}$-states within $\mathcal{A}'$'s states visit α. ◀

▶ **Theorem 24.** *The translation of a GFG ABW or GFG ACW $\mathcal{A}$ to an equivalent DBW or DCW, respectively, involves a $2^{\Theta(n)}$ state blow-up.*

7 Conclusions

Alternating GFG is the sum of its parts. Throughout our five definitions of GFG for alternating automata, a common theme prevails: each definition can be divided into a condition for nondeterminism and one for universality, and their conjunction guarantees good-for-gameness. For example, it suffices for an automaton to compose with both universal automata and nondeterministic automata for it to compose with alternating automata, even alternating tree automata. In other words, GFG nondeterminism and universality cannot interact pathologically to generate alternating automata that are not GFG, and neither can they ensure GFGness without each being GFG independently. This should facilitate checking GFGness, as it can be done separately for universality and nondeterminism.

Between words, trees, games, and automata. In the recent translations from alternating parity word automata into weak automata [7, 16], the key techniques involve adapting methods that use *finite one-player games* to process *infinite* structures that are in some sense between words and trees, and use these to manipulate alternating automata. These translations implicitly depend on the compositionality that enable the step from asymmetrical one-player games, i.e. trees, to alternating automata. Studying good-for-gameness provides us with new tools to move between words, trees, games, and automata, and better understand how nondeterminism, universality, and alternations interact in this context.

References

1 Mikołaj Bojanczyk and Wojciech Czerwinski. An automata toolbox. Technical report, University of Warsaw, 2018.

2 Udi Boker. Why These Automata Types? In *Proceedings of LPAR*, pages 143–163, 2018.

3 Udi Boker, Denis Kuperberg, Orna Kupferman, and Michał Skrzypczak. Nondeterminism in the Presence of a Diverse or Unknown Future. In *Proceedings of ICALP*, pages 89–100, 2013.

4 Udi Boker, Orna Kupferman, and Michał Skrzypczak. How Deterministic are Good-For-Games Automata? In *Proceedings of FSTTCS*, pages 18:1–18:14, 2017.

5 Udi Boker, Orna Kupferman, and Adin Rosenberg. Alternation Removal in Büchi Automata. In *Proceedings of ICALP*, pages 76–87, 2010.

6 Udi Boker and Karoliina Lehtinen. Register Games. Submitted to Logical Methods in Computer Science.

7 Udi Boker and Karoliina Lehtinen. On the way to alternating weak automata. In *Proceedings of FSTTCS*, 2018.

8 Udi Boker and Karoliina Lehtinen. Good for Games Automata: From Nondeterminism to Alternation. arXiv, 2019. `arXiv:1906.11624`.

9 J Richard Büchi and Lawrence H Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

10 Cristian S Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Proceedings of STOC*, pages 252–263, 2017.

11 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Proceedings of ICALP*, pages 139–150, 2009.

12 Thomas Colcombet. Forms of Determinism for Automata. In *Proceedings of STACS*, pages 1–23, 2012.

13 Thomas Colcombet. Fonctions régulières de coût. *Habilitation à diriger les recherches, École Doctorale de Sciences Mathématiques de Paris Centre*, 2013.

14 Thomas Colcombet and Nathanaël Fijalkow. Universal graphs and Good for small games automata: New tools for infinite duration games. to appear in proc. of FOSSACS 2019.

15 Wojciech Czerwiński, Laure Daviaud, Nathanaël Fijalkow, Marcin Jurdzińskit, Ranko Lazić, and Pawel Parys. Universal trees grow inside separating automata: Quasi-polynomial lower bounds for parity games. In *Proceedings of SODA*, pages 2333–2349, 2019.

16 Laure Daviaud, Marcin Jurdzinski, and Karoliina Lehtinen. Alternating Weak Automata from Universal Trees. Preprint arXiv, 2019. `arXiv:1903.12620`.

17 Thomas Henzinger and Nir Piterman. Solving games without determinization. In *Proceedings of CSL*, pages 395–410, 2006.

18 Denis Kuperberg and Michał Skrzypczak. On Determinisation of Good-For-Games Automata. In *Proceedings of ICALP*, pages 299–310, 2015.

19 Orna Kupferman, Shmuel Safra, and Moshe Y Vardi. Relating word and tree automata. *Ann. Pure Appl. Logic*, 138(1-3):126–146, 2006. Conference version in 1996.

20 Karoliina Lehtinen. A modal μ perspective on solving parity games in quasi-polynomial time. In *Proceedings of LICS*, pages 639–648, 2018.

21 Satoru Miyano and Takeshi Hayashi. Alternating Finite Automata on ω-Words. *Theoretical Computer Science*, 32:321–330, 1984.

22 Gila Morgenstern. Expressiveness results at the bottom of the ω-regular hierarchy. M.Sc. Thesis, The Hebrew University, 2003.

Determinacy in Discrete-Bidding Infinite-Duration Games

Milad Aghajohari
Sharif University of Technology, Iran
mi.aghajohari@gmail.com

Guy Avni
IST Austria, Klosterneuburg, Austria
guy.avni@ist.ac.at

Thomas A. Henzinger
IST Austria, Klosterneuburg, Austria
tah@ist.ac.at

Abstract

In two-player games on graphs, the players move a token through a graph to produce an infinite path, which determines the winner of the game. Such games are central in formal methods since they model the interaction between a non-terminating system and its environment. In bidding games the players bid for the right to move the token: in each round, the players simultaneously submit bids, and the higher bidder moves the token and pays the other player. Bidding games are known to have a clean and elegant mathematical structure that relies on the ability of the players to submit arbitrarily small bids. Many applications, however, require a fixed granularity for the bids, which can represent, for example, the monetary value expressed in cents. We study, for the first time, the combination of *discrete-bidding* and *infinite-duration* games. Our most important result proves that these games form a large determined subclass of concurrent games, where *determinacy* is the strong property that there always exists exactly one player who can guarantee winning the game. In particular, we show that, in contrast to non-discrete bidding games, the mechanism with which tied bids are resolved plays an important role in discrete-bidding games. We study several natural tie-breaking mechanisms and show that, while some do not admit determinacy, most natural mechanisms imply determinacy for every pair of initial budgets.

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

Keywords and phrases Bidding games, Richman games, determinacy, concurrent games, discrete bidding

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.20

Related Version `https://arxiv.org/abs/1905.03588`

Funding This research was supported in part by the Austrian Science Fund (FWF) under grants S11402-N23 (RiSE/SHiNE), Z211-N23 (Wittgenstein Award), and M 2369-N33 (Meitner fellowship).

1 Introduction

Two-player infinite-duration games on graphs are a central class of games in formal verification [4] and have deep connections to foundations of logic [36]. They are used to model the interaction between a system and its environment, and the problem of synthesizing a correct system then reduces to finding a winning strategy in a graph game [35]. A graph game proceeds by placing a token on a vertex in the graph, which the players move throughout the graph to produce an infinite path ("play") π. The winner of the game is determined according to π.

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 20; pp. 20:1–20:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Two ways to classify graph games are according to the type of *objectives* of the players, and according to the *mode of moving* the token. For example, in *reachability games*, the objective of Player 1 is to reach a designated vertex t, and the objective of Player 2 is to avoid t. An infinite play π is winning for Player 1 iff it visits t. The simplest mode of moving is *turn based*: the vertices are partitioned between the two players and whenever the token reaches a vertex that is controlled by a player, he decides how to move the token.

In *bidding* games, in each turn, a bidding takes place to determine which player moves the token. Bidding games were introduced in [25, 26], where the main focus was on a concrete bidding rule, called *Richman* rule (named after David Richman), which is as follows: Each player has a budget, and before each move, the players simultaneously submit bids, where a bid is legal if it does not exceed the available budget. The player who bids higher wins the bidding, pays the bid to other player, and moves the token.

Bidding games exhibit a clean and elegant theory. The central problem that was previously studied concerned the existence of a necessary and sufficient *threshold budget*, which allows a player to achieve his objective. Assuming the sum of budgets is 1, the threshold budget at a vertex v, denoted $\texttt{Thresh}(v)$, is such that if Player 1's budget exceeds $\texttt{Thresh}(v)$, he can win the game, and if Player 2's budget exceeds $1 - \texttt{Thresh}(v)$, he can win the game. Threshold budgets are known to exist in bidding reachability games [25, 26], as well as infinite-duration bidding games with Richman bidding [7], *poorman* bidding [8], which are similar to Richman bidding except that the winner of a bidding pays the "bank" rather than the other player, and *taxman* bidding [10], which span the spectrum between Richman and poorman bidding. In addition, bidding games exhibit a rich mathematical structure in the form of a connection with *random-turn based* games, which are a special case of *stochastic games* [19] in which in each turn, the player who moves is chosen according to a probability distribution. Random-turn based games have been extensively studied since the seminal paper [34].

These theoretical properties of bidding games highly depend on the ability of the players to submit arbitrarily small bids. Indeed, in poorman games, the bids tend to 0 as the game proceeds. Even in Richman reachability games, when the budget of Player 1 at v is $\texttt{Thresh}(v) + \epsilon$, a winning strategy bids so that the budget always exceeds the threshold budget and, either the game is won or Player 1's surplus, namely the difference between his budget and the threshold budget, strictly increases. This strategy uses bids that are exponentially smaller than ϵ.

For practical applications, however, allowing arbitrary granularity of bids is unreasonable. For example, in formal methods, graph games are used to reason about multi-process systems, and bidding naturally models "scrip" systems, which use internal currency in order to prioritize processes. Car-control systems are one example, where different components might send conflicting actions to the engine, e.g., the cruise control component can send the action "accelerate" while the traffic-light recognizer can send "stop". Bidding then specifies the level of criticality of the actions, yet for this mechanism to be practical, the number of levels of criticality (bids) must stay small. Bidding games can be used in settings in which bids represent the monetary value of choosing an action. Such settings typically have a finite granularity, e.g., cents. One such setting is Blockchain technology [15, 5], where players represent agents that are using the service, and their bids represent transaction fees to the miners. A second such setting is reasoning about ongoing auctions like the ones used in the internet for advertisement allocation [32]. Bidding games can be used to devise bidding strategies in such auctions. Motivation for bidding games also comes from recreational games, e.g., bidding chess [12] or tic-tac-toe[1], where it is unreasonable for a human player to keep track of arbitrarily small and possibly irrational numbers.

In this work, we study *discrete-bidding games* in which the granularity of the bids is restricted to be natural numbers. A key difference from the continuous-bidding model is that there, the issue of how to break ties was largely ignored, by only considering cases where the initial budget does not equal $\texttt{Thresh}(v)$. In discrete-bidding, however, ties are a central part of the game. A discrete-bidding game is characterized explicitly by a tie-breaking mechanism in addition to the standard components, i.e., an arena, the players' budgets, and an objective. We investigate several tie-breaking mechanisms and show how they affect the properties of the game. Discrete-bidding games with reachability objectives were first studied in [20]. The focus in that paper was on extending the Richman theory to the discrete domain, and we elaborate on their results later in this section.

A central concept in game theory is a *winning strategy*: a strategy that a player can reveal before the other player, and still win the game. A game is *determined* if exactly one of the players can guarantee winning the game. The simplest example of a non-determined game is a two-player game called *matching pennies*: Each player chooses 1 ("heads") or 0 ("tails"), and Player 1 wins iff the parity of the sum of the players' choices is 0. Matching pennies is not determined since if Player 1 reveals his choice first, Player 2 will choose opposite and win the game, and dually for Player 2.

Discrete-bidding games are a subclass of *concurrent* graph games [2], in which in each turn, the players simultaneously select actions, and the joint vector of actions determines the next position. A bidding game $\mathcal{G}$ is equivalent to a concurrent game $\mathcal{G}'$ that is played on the "configuration graph" of $\mathcal{G}$: each vertex of $\mathcal{G}'$ is a tuple $\langle v, B_1, B_2, s\rangle$, where v is the vertex in $\mathcal{G}$ on which the token is situated, the players' budgets are B_1 and B_2, and s is the state of the tie-breaking mechanism. An action in $\mathcal{G}'$ corresponds to a bid and a vertex to move to upon winning the bidding. Concurrent games are not in general determined since matching pennies can be modelled as a concurrent game.

The central question we address in this work asks under which conditions bidding games are determined. We show that determinacy in bidding games highly depends on the tie-breaking mechanism under use. We study natural tie-breaking mechanisms, show that some admit determinacy while others do not. The simplest tie-breaking rule we consider alternates between the players: Player 1 starts with the *advantage*, when a tie occurs, the player with the advantage wins, and the advantage switches to the other player. We show that discrete-bidding games with alternating tie-breaking are not determined, as we demonstrate below.

▶ **Example 1.** Consider the bidding reachability game that is depicted in Fig. 1. We depict the player who has the advantage with a star. We claim that no player has a winning strategy when the game starts from the configuration $\langle v_0, 1, 1^*\rangle$, thus the token is placed on v_0, both budgets equal 1, and Player 2 has the tie-breaking advantage. We start by showing that if Player 2 reveals his first bid before Player 1, then Player 1 can guarantee winning the game. There are two cases. First, if Player 2 bids 0, Player 1 bids 1 and draws the game to t. Second, if Player 2 bids 1, then Player 1 bids 0, and the game reaches the configuration $\langle v_1, 2, 0^*\rangle$. Next, both players bid 0 and we reach $\langle v_2, 2^*, 0\rangle$. Player 1 wins by bidding 1

[1] http://biddingttt.herokuapp.com/

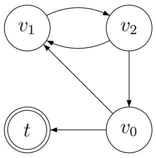


Figure 1 A bidding game that is not determined with alternating tie-breaking, when the initial configuration is $\langle v_0, 1, 1^* \rangle$.

twice; indeed, the next two configurations are $\langle v_0, 1^*, 1\rangle$ and either $\langle t, 0, 2^*\rangle$, if Player 2 bids 1, or $\langle t, 0^*, 2\rangle$, if he bids 0. The proof that Player 1 loses when he reveals his first bid before Player 2 can be found in Theorem 10. ⌟

We generalize the alternating tie-breaking mechanism as follows. A *transducer* is similar to an automaton only that the states are labeled by output letters. In *transducer-based* tie breaking, a transducer is run in parallel to the game. The transducer reads information regarding the biddings and outputs which player wins in case of a tie. Alternating tie-breaking is a special case of transducer tie-breaking in which the transducer is a two-state transducer, where the alphabet consists of the letters $\top$ ("tie") and $\perp$ ("no-tie") and the transducer changes its state only when the first letter is read.

▶ **Example 2.** We describe another simpler game that is not determined. In a *Büchi game*, Player 1 wins a play iff it visits an accepting state infinitely often. We claim that the Büchi bidding game that is depicted on the left of Fig. 2 is not determined when the tie-breaking uses the transducer on the right of the figure and both of the players' initial budgets are positive. That is, if a tie occurs in the first bidding, Player 2 wins all ties for the rest of the game, and otherwise Player 1 wins all ties. First note that, for $i \in \{1, 2\}$, no matter what the budgets are, if Player i wins all ties, he wins the game. A winning strategy for Player i always bids 0. Intuitively, the other player must invest a unit of budget for winning a bidding and leaving v_i, thus the game eventually stays in v_i. So, the winner is determined according to the outcome of the first bidding, and the players essentially play a matching-pennies game in that round. ⌟

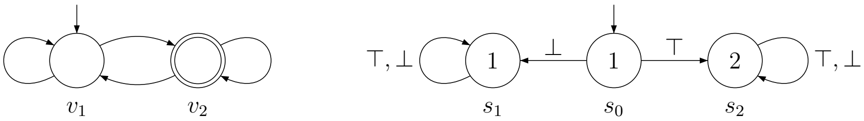


Figure 2 On top, a Büchi game that is not determined when tie-breaking is determined according to the transducer on the bottom, where the letters $\top$ and $\perp$ respectively represent "tie" and "no tie".

We proceed to describe our positive results. For transducer-based tie-breaking, we identify a necessary and sufficient condition for determinacy: when the transducer is un-aware of the occurrence of ties, bidding games are determined. The second tie-breaking mechanism for which we show determinacy is *random tie-breaking*: a tie is resolved by tossing a coin that determines the winner of the bidding. Finally, a tie-breaking mechanism that was introduced in [20] is advantage based, except that when a tie occurs, the player with the advantage can choose between (1) winning the bidding and passing the advantage to the other player, or (2) allowing the other player to win the bidding and keeping the advantage. Determinacy for reachability games with this tie-breaking mechanism was shown in [20]. The technique that is used there cannot be extended to the other tie-breaking mechanisms we study. We show an alternative proof for advantage-based tie-breaking and extend the determinacy result for richer objectives beyond reachability.

We obtain our positive results by developing a unified proof technique to reason about bidding games, which we call *local determinacy*. Intuitively, a concurrent game is locally determined if from each vertex, there is a player who can reveal his action before the other player. We show that locally-determined reachability games are determined and then extend to *Müller* games, which are richer qualitative games. We expect our technique to extend to show determinacy in other fragments of concurrent games unlike the technique in [20], which is tailored for bidding games.

Determinacy has computational complexity implications; namely, finding the winner in a bidding game with objective α when the budgets are given in unary is as hard as solving a turn-based game with objective α, and we show a simple reduction in the other way for bidding games. Finally, we establish results for strongly-connected discrete-bidding games.

Due to lack of space, some proofs appear in the full version.

2 Preliminaries

2.1 Concurrent and turn-based games

A *concurrent* game is a two-player game that is played by placing a token on a graph. In each turn, both players simultaneously select actions, and the next vertex the token moves to is determined according to their choices. The players thus produce an infinite path π in the graph. A game is accompanied by an objective for Player 1, who wins iff π meets his objective. We specify standard objectives in games later in the section. For $i \in \{1, 2\}$, we use $-i$ to refer to the other player, namely $-i = 3 - i$

Formally, a concurrent game is played on an arena $\langle A, V, \lambda, \delta\rangle$, where A is a finite set of actions, V is a finite set of vertices, the function $\lambda : V \times \{1, 2\} \to 2^A \setminus \emptyset$ specifies the allowed actions for Player i in vertex v, and $\delta : V \times A \times A \to V$ specifies, given the current vertex and a choice of actions for the two players, the next vertex the token moves to. We call $u \in V$ a *neighbor* of $v \in V$ if there is $a^1, a^2 \in A$ with $u = \delta(v, a^1, a^2)$. We say that Player i *controls* a vertex $v \in V$ if his actions uniquely determine where the token proceeds to from v. That is, for every $a \in \lambda(v, i)$ there is a vertex u such that, for every allowed action a' of Player $-i$, we have $\delta(v, a, a') = u$. A *turn-based* game is a special case of a concurrent game in which each vertex is controlled by one of the players.

2.2 Bidding games

A (discrete) *bidding game* is a special case of a concurrent game. The game is played on a graph and both players have budgets. In each turn, a bidding takes place to determine which player gets to move the token. Formally, a bidding game is played on an arena $\langle V, E, N, \mathcal{M}\rangle$, where V is a set of vertices, $E \subseteq (V \times V)$ is a set of edges, $N \in \mathbb{N}$ represents the total budget, and the *tie-breaking mechanism* is $\mathcal{M}$ on which we elaborate below.

We formalize the semantics of a bidding game $\langle V, E, N, \mathcal{M}\rangle$ by means of a concurrent game $\langle A, V', \lambda, \delta\rangle$. Let $B_1, B_2 \in \{0, \ldots, N\}$ with $B_1 + B_2 = N$ and $v \in V$. A *configuration vertex* of the form $\langle v, B_1, B_2, s\rangle$, represents a configuration of the bidding game in which the token is placed on v, Player 1's budget is B_1, Player 2's budget is B_2, and s represents the state of the tie-breaking mechanism. The allowed actions in a configuration vertex $\langle v, B_1, B_2, s\rangle$ are $\{0, \ldots, B_1\}$ for Player 1 and $\{0, \ldots, B_2\}$ for Player 2. For bids $b_1, b_2 \in \{0, \ldots, N\}$, the neighbor of a configuration vertex $c = \langle v, B_1, B_2, s\rangle$ is an intermediate vertex $\langle c, b_1, b_2\rangle$. If $b_1 > b_2$, then Player 1 wins the bidding and chooses the next vertex the token proceeds to. In this case, Player 1 controls $\langle c, b_1, b_2\rangle$ and its neighbors are configuration vertices of the form $\langle v', B_1 - b_1, B_2 + b_1, s'\rangle$, where $v' \in V$ with $\langle v, v'\rangle \in E$ and s' is the updated tie-breaking state. The case where Player 2 wins the bidding, i.e., $b_1 < b_2$, is dual.

We proceed to the case of ties and describe three types of tie-breaking mechanisms.

- **Transducer-based:** A *transducer* is $T = \langle \Sigma, Q, q_0, \Delta, \Gamma\rangle$, where Σ is a set of letters, Q is a set of states, $q_0 \in Q$ is an initial state, $\Delta : Q \times \Sigma \to Q$ is a deterministic transition function, and $\Gamma : Q \to \{1, 2\}$ is a labeling of the states. Intuitively, T is run in parallel to the bidding game and its state is updated according to the outcomes of the biddings. Whenever a tie occurs and T is in state $s \in Q$, the winner of the bidding is $\Gamma(s)$. The information according to which tie-breaking is determined is represented by the alphabet of T. In general, the information can include the vertex on which the token is located and the result of the previous bidding, i.e., the winner, whether or not a tie occurred, and the winning bid, thus $\Sigma = V \times \{1, 2\} \times \{\bot, \top\} \times \mathbb{N}$.
- **Random-based:** A tie is resolved by choosing the winner uniformly at random.
- **Advantage-based:** Exactly one player holds the *advantage*. Suppose Player i holds the advantage and a tie occurs. Then Player i chooses who wins the bidding. If he calls the other player the winner, Player i keeps the advantage, and if he calls himself the winner, the advantage switches to the other player.

Formally, consider a configuration $c = \langle v, B_1, B_2, s\rangle$ and an intermediate vertex $\langle c, b_1, b_2\rangle$. In the transducer-based mechanism, the state s is a state in the transducer T. If $b_1 \neq b_2$, the player who controls $\langle v, b_1, b_2\rangle$ is determined as in the above. In case $b_1 = b_2$, then Player $\Gamma(s)$ controls the vertex. In both cases, we update the state of the tie-breaking mechanism by feeding it the information on the last bidding; who won, whether a tie occurred, and what vertex the winner chose, thus we set $s' = \Delta(s, \sigma)$, where $\sigma = \langle v', i, \bot, b_i\rangle$ in case Player i wins the bidding with his bid of b_i, moves to v', and no tie occurs. The other cases are similar.

In random-based tie-breaking, the mechanism has no state, thus we can completely omit s. Consider an intermediate vertex $\langle c, b_1, b_2\rangle$. The case of $b_1 \neq b_2$ is as in the above. Suppose both players bid b. The intermediate vertex $\langle c, b, b\rangle$ is controlled by "Nature". It has two probabilistic outgoing transitions; one transition leads to the intermediate vertex $\langle c, b, b - 1\rangle$, which represents Player 1 winning the bidding with a bid of b, and the other to the intermediate vertex $\langle c, b - 1, b\rangle$, which represents Player 2 winning the bidding with a bid of b. We elaborate on the semantics of concurrent games with probabilistic edges in Section 5.

Finally, in advantage-based tie-breaking, the state of the mechanism represents which player has the advantage, thus $s \in \{1, 2\}$. Consider an intermediate vertex $\langle c, b_1, b_2\rangle$. When a tie does not occur, there is no need to update s. When $b_1 = b_2$, then Player s controls $\langle c, b_1, b_2\rangle$ and the possibility to choose who wins the bidding. Choosing to lose the bidding is modelled by no update to s and moving to an intermediate vertex that is controlled by Player $-s$ from which he chooses a successor vertex and the budgets are updated accordingly. When Player s chooses to win the bidding we proceed directly to the next configuration vertex, update the budgets, and the mechanism's state to $3 - s$.

2.3 Strategies, plays, and objectives

A *strategy* is, intuitively, a recipe that dictates the actions that a player chooses in a game. Formally, a finite *history* of a concurrent game is a sequence $\langle v_0, a^1_0, a^2_0\rangle, \ldots, \langle v_{n-1}, a^1_{n-1}, a^2_{n-1}\rangle, v_n \in (V \times A \times A)^* \cdot V$ such that, for each $0 \leq i < n$, we have $v_{i+1} = \delta(v_i, a^1_i, a^2_i)$. A strategy is a function from $(V \times A \times A)^* \cdot V$ to A. We restrict attention to legal strategies that assign only allowed actions, thus for every history $\pi \in (V \times A \times A)^* \cdot V$ that ends in $v \in V$, a legal strategy σ_i for Player i has $\sigma_i(\pi) \in \lambda(v,i)$. Two strategies σ_1 and σ_2 for the two players and an initial vertex v_0, determine a unique *play*, denoted $\text{play}(v_0,\sigma_1,\sigma_2) \in (V \times A \times A)^\omega$, which is defined as follows. The first element of $\text{play}(v_0,\sigma_1,\sigma_2)$ is $\langle v_0, \sigma_1(v_0), \sigma_2(v_0)\rangle$. For $i \geq 1$, let π^i denote the prefix of length i of $\text{play}(v_0,\sigma_1,\sigma_2)$ and suppose its last element is $\langle v_i, a_i^1, a_i^2\rangle$. We define $v_{i+1} = \delta(v_i, a_1^i, a_2^i)$, $a_1^{i+1} = \sigma_1(\pi^i \cdot v_{i+1})$, and $a_2^{i+1} = \sigma_2(\pi^i \cdot v_{i+1})$. The *path* that corresponds to $\text{play}(v_0,\sigma_1,\sigma_2)$ is $v_0, v_1, \ldots$.

An *objective* for Player 1 is a subset on infinite paths $\alpha \subseteq V^\omega$. We say that Player 1 wins $\text{play}(v_0,\sigma_1,\sigma_2)$ iff the path π that corresponds to $\text{play}(v_0,\sigma_1,\sigma_2)$ satisfies the objective, i.e., $\pi \in \alpha$. Let $inf(\pi) \subseteq V$ be the subset of vertices that π visits infinitely often. We consider the following objectives.

- **Reachability:** A game is equipped with a target set $T \subseteq V$. A play π is winning for Player 1, the reachability player, iff it visits T.
- **Büchi:** A game is equipped with a set $T \subseteq V$ of accepting vertices. A play π is winning for Player 1 iff it visits T infinitely often.
- **Parity:** A game is equipped with a function $p : V \to \{1,\ldots,d\}$, for $d \in \mathbb{N}$. A play π is winning for Player 1 iff $\max_{v \in inf(\pi)} p(v)$ is odd.
- **Müller:** A game is equipped with a set $T \subseteq 2^V$. A play π is winning for Player 1 iff $inf(\pi) \in T$.

3 A Framework for Proving Determinacy

3.1 Determinacy

Determinacy is a strong property of games, which intuitively says that exactly one player has a winning strategy. That is, the winner can reveal his strategy before the other player, and the loser, knowing how the winner plays, still loses. Formally, a strategy σ_i is a *winning strategy* for Player i at vertex v iff for every strategy σ_{-i} for Player $-i$, Player i wins $\text{play}(v,\sigma_1,\sigma_2)$. We say that a game $\langle V, E, \alpha\rangle$ is *determined* if from every vertex $v \in V$ either Player 1 has a winning strategy from v or Player 2 has a winning strategy from v.

While concurrent games are not determined (e.g., “matching pennies”), turn-based games are largely determined.

▶ **Theorem 3** ([28]). *Turn-based games with objectives that are Borel sets are determined. In particular, turn-based Müller games are determined.*

We describe an alternative definition for determinacy in concurrent games. Consider a concurrent game $\mathcal{G} = \langle A, V, \lambda, \delta, \alpha\rangle$. Recall that in $\mathcal{G}$, in each turn, the players simultaneously select an action, and their joint actions determine where the token moves to. For $i \in \{1,2\}$, let $\mathcal{G}_i$ be the turn-based game that, assuming the token is placed on a vertex v, Player i selects an action first, then Player $-i$ selects an action, and the token proceeds from v as in $\mathcal{G}$ given the two actions. Formally, the game $\mathcal{G}_1$ is a turn-based game $\langle A, V \cup (V \times A), \lambda', \delta', \alpha'\rangle$, and the definition for $\mathcal{G}_2$ is dual. The vertices that are controlled by Player 1 are $V_1 = V$ and $V_2 = V \times A$. For $v \in V$, we have $\lambda'(v,1) = \lambda(v,1)$ and since Player 1 controls v, we arbitrarily fix $\lambda'(v,2) = A$. For $a_1 \in \lambda(v,1)$ and $a_2 \in A$, we define $\delta(v,a_1,a_2) = \langle v, a_1\rangle$. Similarly, we define $\lambda'(\langle v,a_1\rangle,1) = A$ and $\lambda'(\langle v,a_1\rangle,2) = \lambda(v,2)$. For $a_1' \in A$ and $a_2 \in \lambda(v,2)$, we define $\delta'(\langle v,a_1\rangle, a_1', a_2) = \delta(v,a_1,a_2)$. Finally, an infinite play $v_1, \langle v_1,a_1\rangle, v_2, \langle v_2,a_2\rangle, \ldots$, is in α' iff $v_1, v_2, \ldots$ is in α. Recall that in bidding games, intermediate vertices are controlled by one player and the only concurrent moves occur when revealing bids. Thus, when $\mathcal{G}$ is a bidding game, in $\mathcal{G}_i$, Player i always reveals his bids before Player $-i$.

▶ **Proposition 4.** *A strategy σ_i is winning for Player i in $\mathcal{G}$ at vertex v iff it is winning in $\mathcal{G}_i$ from v. Then, $\mathcal{G}$ is determined at v iff either Player 1 wins in $\mathcal{G}_1$ from v or Player 2 wins in $\mathcal{G}_2$ from v.*

3.2 Local and global determinacy

We define *local determinacy* in a fragment of concurrent games, which slightly generalizes bidding games. Consider a transducer $R = \langle A \times A, Q, q_0, \Delta, \Gamma \rangle$, where $\Delta : Q \times A \times A \to Q$ is a partial function. We assume that for each state $q \in Q$ and $i \in \{1, 2\}$, there is a set of *allowed actions* for each player, given by $\lambda_R : Q \times \{1, 2\} \to 2^A \setminus \{\emptyset\}$. For each $a_1 \in \lambda_R(q, 1)$ and $a_2 \in \lambda_R(q, 2)$ we require that $\Delta(q, a_1, a_2)$ is defined. Recall that $\Gamma : Q \to \{1, 2\}$.

We say that a concurrent game $\mathcal{G} = \langle A, V, \lambda, \delta \rangle$ with objective α is *R-concurrent* if (1) the set of vertices V are partitioned into *configuration* vertices C and *intermediate* vertices I, (2) intermediate vertices do not contribute to the objective, thus for two plays π and π' that differ only in their intermediate vertices, we have $\pi \in \alpha$ iff $\pi' \in \alpha$, (3) the neighbors of configuration vertices are intermediate vertices and the transition function restricted to configuration vertices is one-to-one, i.e., for every configuration vertex c and two pairs of actions $\langle a_1, a_2 \rangle \neq \langle a_1', a_2' \rangle$, we have $\delta(c, a_1, a_2) \neq \delta(c, a_1', a_2')$, (4) each intermediate vertex is controlled by one player and its neighbors can either be all intermediate or all configuration vertices, (5) for $v, v' \in V$ such that $N(v), N(v') \subseteq I$, we have $N(v) \cap N(v') = \emptyset$, (6) each vertex in V is associated with a state in R with the following restrictions. Suppose $c \in C$ is associated with $q \in Q$. Then, $\lambda(v, i) = \lambda_R(q, i)$, for $i \in \{1, 2\}$. The transducer updates its state after concurrent moves in configuration vertices; namely, for a configuration vertex c and two actions $a_1, a_2 \in A$, let $u = \delta(c, a_1, a_2)$ be an intermediate vertex. Then, the state that is associated with u is $q' = \Delta(q, a_1, a_2)$ and u is controlled by Player $\Gamma(q')$. The transducer also updates its state between intermediate states; namely, if $u' \in I$ is a neighbor of u and assume Player 1 controls u and chooses action a_1 to proceed from u to u', then u' is associated with $\Delta(q', a_1, a_2)$, for all $a_2 \in A$, and similarly for Player 2. Finally, the transducer does not update its state when proceeding from an intermediate vertex to a configuration one; namely, if $c' \in C$ is a neighbor of $u \in I$ and u is associated with $q \in Q$, then c' is associated with q.

Each bidding game with transducer- and advantage-based tie-breaking is R-concurrent. Indeed, suppose the sum of budgets is $N \in \mathbb{N}$ in the bidding game. Then, the states of the transducer model the players' budget and the state of the tie-breaking mechanism. Thus, each state of the transducer is a triple $\langle B_1, B_2, s \rangle$ such that $B_1 + B_2 = N$. The set of allowed actions in a state $\langle B_1, B_2, s \rangle$ are the allowed bids, thus $\lambda_R(\langle B_1, B_2, s \rangle, i) = \{0, \ldots, B_i\}$, for $i \in \{1, 2\}$. Following a bidding in a configuration vertex, the intermediate vertex is obtained similarly to bidding games; namely, the budgets are updated by reducing the winning bid from the winner's budget and adding it to the loser's budget, and the state of the tie-breaking mechanism is updated. With transducer-based tie-breaking, we need only one intermediate vertex between two configuration vertices since we use the information from the bidding to update the state of the tie-breaking transducer. In advantage-based tie-breaking, when no tie occurs, a single intermediate vertex is needed since there is no update to the state of the tie-breaking mechanism. In case of a tie, however, a second intermediate vertex is needed in order to allow the player who holds the advantage, the chance to decide whether or not to use it.

We describe the intuition for local determinacy. Consider a concurrent game $\mathcal{G}$ and a vertex v. Recall that it is generally not the case that $\mathcal{G}$ is determined. That is, it is possible that neither Player 1 nor Player 2 have a winning strategy from v. Suppose Player 1 has no winning strategy. We say that a transducer admits local determinacy if in every vertex v that is not winning for Player 1, there is a Player 2 action that he can reveal before Player 1 and stay in a non-losing vertex. Formally, we have the following.

▶ **Definition 5** (Local determinacy). *We say that a transducer R admits local determinacy if every concurrent game $\mathcal{G}$ with Borel objective that is R-concurrent has the following property. Consider the turn-based game $\mathcal{G}_1$ in which Player 1 reveals his action first in each position. Since α is Borel, it is a determined game and there is a partition of the vertices to losing and winning vertices for Player 1. Then, for every vertex $v \in V$ that is losing for Player 1 in $\mathcal{G}_1$, there is a Player 2 action a_2 such that, for every Player 1 action a_1, the vertex $\delta(v, a_1, a_2)$ is losing for Player 1 in $\mathcal{G}_1$.*

We show that locally-determined games are determined by starting with reachability objectives and working our way up to Müller objectives.

▶ **Lemma 6.** *If a reachability game $\mathcal{G}$ is R-concurrent for a locally-determined transducer R, then $\mathcal{G}$ is determined.*

Proof. Consider a concurrent reachability game $\mathcal{G} = \langle A, V, \lambda, \delta, \alpha \rangle$ and a vertex $v \in V$ from which Player 1 does not have a winning strategy. That is, v is losing for Player 1 in $\mathcal{G}_1$. We describe a winning strategy for Player 2 from v in $\mathcal{G}$. Player 2's strategy maintains the invariant that the set of vertices S that are visited along the play in $\mathcal{G}$, are losing for Player 1 in $\mathcal{G}_1$. Recall that since we assume intermediate vertices do not contribute to the objective, the target of Player 1 is a configuration vertex. The invariant implies that Player 2 wins since there is no intersection between S and Player 1's target, and thus the target is never reached. Initially, the invariant holds by the assumption that v is losing for Player 1 in $\mathcal{G}_1$. Suppose the token is placed on a vertex u in $\mathcal{G}$. Local determinacy implies that Player 2 can choose an action a_2 that guarantees that no matter how Player 1 chooses, the game reaches a losing vertex for Player 1 in $\mathcal{G}_1$. Thus, the invariant is maintained, and we are done. ◀

Next, we show determinacy in parity games by reducing them to reachability games. The reduction relies on a well-known concept that is called *cycle-forming game* (see for example [3]) in which we terminate the parity game once a cycle is formed. Given a parity game $\mathcal{P}$ and a vertex v in $\mathcal{P}$, for $i \in \{1, 2\}$, by definition, Player i wins in $\mathcal{P}$ from v iff he wins from v in $\mathcal{P}_i$, in which he reveals his bids first. Memoryless determinacy of turn-based parity games [21] implies that Player 1 wins from v in $\mathcal{P}_i$ iff he wins from v in the cycle-forming game $CFG(\mathcal{P}_i, v)$. By applying the cycle-forming game construction directly to an R-concurrent game $\mathcal{P}$, we obtain a reachability game $CFG(\mathcal{P}, v)$ that is R-concurrent, which, by Lemma 6 is determined. It is technical to show that Player i wins from v in $CFG(\mathcal{P}_i, v)$ iff he wins from v in $CFG(\mathcal{P}, v)_i$. Thus, if Player 1 does not win from v in $\mathcal{P}$, Player 2 wins from v. The details of the proof of the following lemma can be found in the full version.

▶ **Lemma 7.** *If a parity game $\mathcal{P}$ is R-concurrent for a locally-determined transducer R, then $\mathcal{P}$ is determined.*

The proof for Müller objectives is similar only that we replace the cycle-forming game reduction with a reduction from Müller games to parity games [23, Chapter 2].

▶ **Theorem 8.** *If a Müller game $\mathcal{G}$ is R-concurrent for a locally-determined transducer R, then $\mathcal{G}$ is determined.*

3.3 The bidding matrix

Consider a bidding game $\mathcal{G} = \langle V, E, N, \mathcal{M}, \alpha \rangle$. Recall that $\mathcal{G}$ is R-concurrent, where a configuration vertex is of the form $c = \langle v, B_1, B_2, s \rangle$. The set of allowed actions in c for Player i is $\{0, \dots, B_i\}$, for $i \in \{1, 2\}$. In particular, there is a natural order on the actions. We think of the possible pairs of actions available in c as a matrix M_c, which we call the *bidding matrix*. Rows in M_c correspond to Player 1 bids and columns corresponds to Player 2 bids. The diagonal that starts in the top-left corner of M_c and follows entries of the form $\langle j, j \rangle$, for $0 \leq j \leq \min\{B_1, B_2\}$, corresponds to biddings that resolve in a tie. Entries above and below it correspond to biddings that are winning for Player 2 and Player 1, respectively. Consider the turn-based game $\mathcal{G}_1$ in which Player 1 reveals his bid first. We consider objectives for which turn-based games are determined, thus in $\mathcal{G}_1$, the vertex $\langle c, b_1, b_2 \rangle$ is either winning for Player 1 or Player 2. The entries in M_c are in $\{1, 2\}$, where $M_c(b_1, b_2) = 1$ iff the intermediate vertex $\langle c, b_1, b_2 \rangle$ is winning for Player 1 in $\mathcal{G}_1$.

For $i \in \{1, 2\}$, we call a row or column in M_c an *i-row* or *i-column*, respectively, if all its entries are i. We rephrase local determinacy in bidding games in terms of the bidding matrix, and it is not hard to show that the following definition implies Definition 5.

▶ **Definition 9.** *Consider a bidding game $\mathcal{G} = \langle V, E, N, \mathcal{M}, \alpha \rangle$. We say that $\mathcal{G}$ is locally determined if for every configuration vertex c, the bidding matrix either has a 2-column or a 1-row.*

4 Transducer-based tie-breaking

The determinacy of bidding games with transducer-based tie-breaking depends on the information that is available to the transducer. We start with a negative result.

▶ **Theorem 10.** *Reachability bidding games with alternate tie-breaking are not determined.*

Proof. Consider the bidding reachability game that is depicted in Fig. 1. We show that no player has a winning strategy when the game starts from the configuration $\langle v_0, 1, 1^* \rangle$, thus the token is placed on v_0, both budgets equal 1, and Player 2 has the tie-breaking advantage. The proof that Player 2 has no winning strategy is shown in Example 1. We show that Player 1 has no winning strategy, thus if he reveals his first bid before Player 2, then Player 2 wins the game. In Fig. 3, we depict most of the relevant configurations in the game with Player 2's strategy in place, and it is not hard to verify that Player 2 can force the game to avoid t. ◀

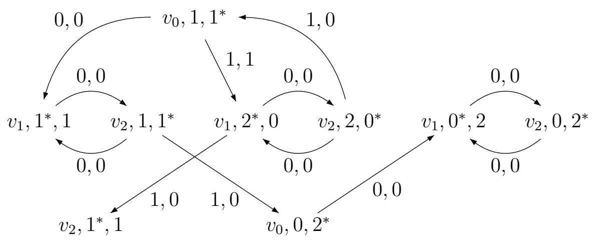


Figure 3 Configurations in the game that is depicted in Fig. 1.

We proceed to prove our positive results, namely that bidding games are determined when the information according to which tie-breaking is determined does not include the occurrence of ties. Formally, we define a subclass of tie-breaking transducers.

▶ **Definition 11.** *A transducer is* un-aware of ties *when its alphabet is $V \times \{1,2\} \times I\!N$, where a letter $\langle v, i, b\rangle \in V \times \{1,2\} \times I\!N$ means that the token is placed on v and Player i wins the bidding with his bid of b.*

We start with the following lemma, whose proof can be found in the full version, that applies to any tie-breaking mechanism. Recall that rows represent Player 1 bids, columns represent Player 2 bids, entries on the top-left to bottom-right diagonal represent ties in the bidding, entries above it represent Player 2 wins, and entries below represent Player 1 wins.

▶ **Lemma 12.** *Consider a bidding game $\mathcal{G}$ with some tie-breaking mechanism T and consider a configuration $c = \langle v, B_1, B_2, s\rangle$. Entries in M_c in a column above the diagonal are all equal, thus for bids $b_2 > b_1, b_1'$, the entries $\langle b_1, b_2\rangle$ and $\langle b_1', b_2\rangle$ in M_c are equal. Also, the entries in a row to the left of the diagonal are equal, thus for bids $b_1 > b_2, b_2'$, the entries $\langle b_1, b_2\rangle$ and $\langle b_1, b_2'\rangle$ in M_c are equal.*

The next lemma, whose proof can be found in the full version, relates an entry on the diagonal with its neighbors.

▶ **Lemma 13.** *Consider a bidding game $\mathcal{G}$, where tie-breaking is resolved according to a transducer T that is un-aware of ties. Consider a configuration $c = \langle v, B_1, B_2, s\rangle$. Let $b \in I\!N$. If $\Gamma(s) = 1$, i.e., Player 1 wins ties in c, then the entries $\langle b, b\rangle$ and $\langle b, b-1\rangle$ in M_c are equal. Dually, if $\Gamma(s) = 2$, then the entries $\langle b, b\rangle$ and $\langle b-1, b\rangle$ in M_c are equal.*

We continue to prove our positive results.

▶ **Theorem 14.** *Consider a tie-breaking transducer T that is un-aware of ties. Then, a Müller bidding game that resolves ties using T is determined.*

Proof. We show that transducers that are not aware of ties admit local determinacy, and the theorem follows from Theorem 8. See a depiction of the proof in Figure 4.

Consider a bidding game $\langle V, E, \alpha, N, T\rangle$, where T is un-aware of ties, and consider a configuration vertex $c = \langle v, B_1, B_2, s\rangle$. We show that M_c either has a 1-row or a 2-column. We prove for $\Gamma(s) = 1$ and the proof for $\Gamma(s) = 2$ is similar. Let $B = \min\{B_1, B_2\}$. When $B_2 > B_1$, the matrix M_c is a rectangle. Still the diagonal of interest models biddings that result in ties and it starts from the top right corner of M_c. The columns $B+1, \ldots, B_2$ do not intersect this diagonal. By Lemma 12, the entries in each one of these columns are all equal. We assume all the entries are 1 as otherwise we find a 2-column. Similarly, if $B_1 > B_2$, we assume that the entries in the rows $B+1, \ldots, B_1$ below the diagonal are all 2, otherwise we find a 1-row.

We restrict attention to the $B \times B$ top-left sub-matrix of M_c. Consider the B-th row in M_c. By Lemma 12, entries in this row that are below the diagonal are all equal, and, since $\Gamma(s) = 1$, they also equal the entry on the diagonal. If all entries equal 1, then together with the assumption above that entries to the right of the diagonal are all 1, we find a 1-row. Thus, we assume all entries below and on the diagonal in the B-th row all equal 2. Now, consider the B-th column. By Lemma 12, the entries above the diagonal are all equal. If they all equal 2, together with the entry $\langle B, B\rangle$ on the diagonal and the entries below it, which we assume are all 2, we find a 2-column. Thus, we assume the entries in the B-th column above the diagonal are all 1. Next, consider the $(B-1)$-row. Similarly, the elements on and to the left of the diagonal are all equal, and if they equal 1, we find a 1-row, thus we assume they are all 2. We continue in a similar manner until the entry $\langle 1, 1\rangle$. If it is 1, we find a 1-column and if it is 2, we find a 2-row, and we are done. ◀

We conclude this section by relating the computational complexity of bidding games with turn-based games. Let TB_α be the class of turn-based games with a qualitative objective α. Let $\mathrm{BID}_{\alpha,\mathrm{trans}}$ be the class of bidding games with transducer-based tie-breaking and objective α. The problem TB-WIN_α gets a game $\mathcal{G} \in \mathrm{TB}_\alpha$ and a vertex v in $\mathcal{G}$, and the goal is to decide whether Player 1 can win from v. Similarly, the problem $\text{BID-WIN}_{\alpha,\mathrm{trans}}$ gets as input a game $\mathcal{G} \in \mathrm{BID}_{\alpha,\mathrm{trans}}$ with budgets expressed in unary and a configuration c in $\mathcal{G}$, and the goal is to decide whether Player 1 can win from c. The proof of the following theorem can be found in the full version.

▶ **Theorem 15.** *For a qualitative objective α, the complexity of $TB\text{-}WIN_\alpha$ and $BID\text{-}WIN_{\alpha,trans}$ coincide.*

5 Random-Based Tie Breaking

In this section we show that bidding games with random-based tie-breaking are determined. A stochastic concurrent game is $\mathcal{G} = \langle A, V, \lambda, \delta, \alpha\rangle$ is the same as a concurrent game only that the transition function is stochastic, thus given $v \in V$ and $a^1, a^2 \in A$, the transition function $\delta(v, a^1, a^2)$ is a probability distribution over V. Two strategies σ_1 and σ_2 give rise to a probability distribution $D(\sigma_1, \sigma_2)$ over infinite plays.

Traditionally, determinacy in stochastic concurrent games states that each vertex is associated with a *value*, which is the probability that Player 1 wins under optimal play [27]. The value is obtained, however, when the players are allowed to use probabilistic strategies. We show a stronger form of determinacy in bidding games; namely, we show that the value exists even when the players are restricted to use deterministic strategies.

▶ **Definition 16** (Determinacy in stochastic games)**.** *Consider a stochastic concurrent game $\mathcal{G}$ and a vertex $v \in V$. Let P_1 and P_2 denote the set of pure strategies for Players 1 and 2, respectively. For $i \in \{1,2\}$, the value for Player i, denoted $val_i(\mathcal{G}, v)$, is intuitively obtained when he reveals his strategy before the other player. We define $val_1(\mathcal{G}, v) = \sup_{\sigma_1\in P_1} \inf_{\sigma_2\in P_2} \Pr_{\pi\sim D(\sigma_1,\sigma_2)}[\pi \in \alpha]$ and $val_2(\mathcal{G}, v) = \inf_{\sigma_2\in P_2} \sup_{\sigma_1\in P_1} \Pr_{\pi\sim D(\sigma_1,\sigma_2)}[\pi \in \alpha]$. We say that $\mathcal{G}$ is determined in v if $val_1(\mathcal{G}, v) = val_2(\mathcal{G}, v)$ in which case we denote the value by $val(\mathcal{G}, v)$. We say that $\mathcal{G}$ is determined if it is determined in all vertices.*

The key idea in the proof shows determinacy for reachability games that are played on directed acyclic graphs (DAGs, for short). The following lemma, whose proof can be found in the full version, shows that the proof for DAGs implies the general case by formalizing a standard "unwinding" argument (see for example Theorem 3.7 in [20]).

▶ **Lemma 17.** *Determinacy of reachability bidding games that are played on DAGs implies determinacy of general reachability bidding games.*

We continue to show determinacy in bidding games on DAGs.

▶ **Lemma 18.** *Reachability bidding games with random-based tie-breaking that are played on DAGs are determined.*

Proof. Consider a reachability game $\mathcal{G}$ that is played on a DAG with two distinguished vertices t_1 and t_2, which are sinks. There are no other cycles in $\mathcal{G}$, thus all plays end either in t_1 or t_2, and, for $i \in \{1,2\}$, Player i wins iff the game ends in t_i. The *height* of $\mathcal{G}$ is the length of the longest path from some vertex to either t_1 or t_2. We prove that $\mathcal{G}$ is determined by induction on its height. For a height of 0, the claim clearly holds since for every $B_1, B_2 \in \mathbb{N}$, the value in t_1 is 1 and the value in t_2 is 0. Suppose the claim holds for games of heights of at most $n-1$ and we prove for games of height n.

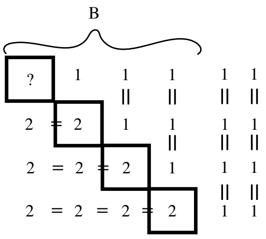


Figure 4 A depiction of the contradiction in Theorem 14 with $B_2 > B_1$.

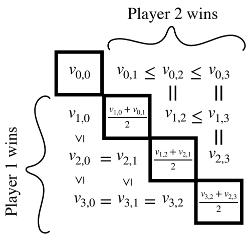

Player 2 wins
Player 1 wins
$v_{0,0}$
$v_{0,1} \leq v_{0,2} \leq v_{0,3}$
$v_{1,0}$
$\frac{v_{1,0}+v_{0,1}}{2}$
$v_{1,2} \leq v_{1,3}$
$v_{2,0} = v_{2,1}$
$\frac{v_{1,2}+v_{2,1}}{2}$
$v_{2,3}$
$v_{3,0} = v_{3,1} = v_{3,2}$
$\frac{v_{3,2}+v_{2,3}}{2}$

Figure 5 Observations on the matrix M_c when resolving ties randomly.

Consider a configuration vertex $c = \langle v, B_1, B_2\rangle$ of height n. Let c' be a configuration vertex that, skipping intermediate vertices, is a neighbor of c. Then, the height of c' is less than n and by the induction hypothesis, its value is well defined. It follows that the value of the intermediate vertices following c are also well-defined: if the intermediate vertex is controlled by Player 1 or Player 2, the value is respectively the maximum or minimum of its neighbors, and if it is controlled by Nature, the value is the average of its two neighbors.

We claim that $\mathcal{G}$ is determined in c by showing that one of the players has a (weakly) *dominant* bid from c, where a bid b_1 dominates a bid b'_1 if, intuitively, Player 1 always prefers bidding b_1 over b'_1. It is convenient to consider a variant of the bidding matrix M_c of c, which is a $(B_1 + 1) \times (B_2 + 1)$ matrix with entries in $[0, 1]$, where an entry $M_c(b_1, b_2)$ represents the value of the intermediate vertex $\langle c, b_1, b_2\rangle$. Note that Player 1, the reachability player, aims to maximize the value while Player 2 aims to minimize it. Some properties of M_c are depicted in Fig. 5 and are formalized in the full version.

Consider the bids 0 and 1 for the two players. We claim that there is a player for which either 0 weakly dominates 1 or vice versa. Assume towards contradiction that this is not the case. Consider the 2×2 top-left sub-matrix of M_c and denote its values $v_{0,0}, v_{0,1}, v_{1,0}$, and $v_{1,1}$. Since $v_{1,1}$ is the average of $v_{0,1}$ and $v_{1,0}$, we either have $v_{0,1} \leq v_{1,1} \leq v_{1,0}$ or $v_{0,1} \geq v_{1,1} \geq v_{1,0}$. Suppose w.l.o.g. that the first holds, thus $v_{0,1} \leq v_{1,0}$. Note that $v_{0,0} < v_{0,1}$, since otherwise the bid 1 dominates 0 for Player 2. Also, we have $v_{0,0} > v_{1,0}$, since otherwise 0 dominates 1 for Player 1. Combining, we have that $v_{0,1} > v_{1,0}$, and we reach a contradiction.

In the full version, we show that (1) if row 0 dominates row 1, it dominates every other row, in which case we can find optimal pure strategies in c, and (2) if row 1 dominates row 0, then column 1 dominates column 0, in which case we can delete the first row and first column and reason about a smaller game. Two dual properties hold for columns. ◀

Combining the two theorems above, we obtain the following.

▶ **Theorem 19.** *Reachability bidding games with random-based tie breaking are determined.*

6 Advantage-Based Tie-Breaking

Recall that in advantage-based tie-breaking, one of the players holds the advantage, and when a tie occurs, he can choose whether to win and pass the advantage to the other player, or lose the bidding and keep the advantage. Advantage-based tie-breaking was introduced and studied in [20], where determinacy for reachability games was obtained by showing that each vertex v in the game has a threshold budget $\texttt{Thresh}(v) \in (\mathbb{N} \times \{*\})$ such that that Player 1 wins from v iff his budget is at least $\texttt{Thresh}(v)$, where $n^* \in (\mathbb{N} \times \{*\})$ means that

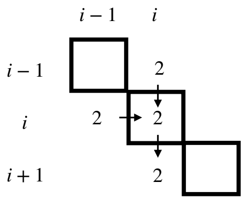


Figure 6 A depiction of the first part of Lemma 21.

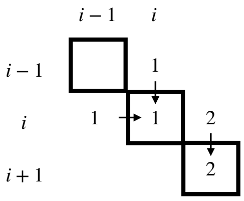


Figure 7 A depiction of the second part of Lemma 21.

Player 1 wins when he starts with a budget of n as well as the advantage. We show that advantage-based tie-breaking admits local determinacy, thus Müller bidding games with advantage-based are determined.

Recall that the state of the advantage-based tie-breaking mechanism represents which player has the advantage, thus it is in $\{1, 2\}$.

▶ **Lemma 20** ([20]). *Consider a reachability bidding game $\mathcal{G}$ with advantage-based tie-breaking.*

- *Holding the advantage is advantageous: For $i \in \{1, 2\}$, if Player i wins from a configuration vertex $\langle v, B_1, B_2, \neg i\rangle$, then he also wins from $\langle v, B_1, B_2, i\rangle$.*
- *The advantage can be replaced by a unit of budget: Suppose Player 1 wins in $\langle v, B_1, B_2, 1\rangle$, then he also wins in $\langle v, B_1 + 1, B_2 - 1, 2\rangle$. Suppose Player 2 wins in $\langle v, B_1, B_2, 2\rangle$, then he also wins in $\langle v, B_1 - 1, B_2 + 1, 1\rangle$.*

We need two more observation on the bidding matrix, which are depicted in Figs. 6 and 7, stated in Lemma 21 below, and proven in the full version.

▶ **Lemma 21.** *Consider a reachability bidding game $\mathcal{G}$ with advantage-based tie-breaking. Consider a configuration $c = \langle v, B_1, B_2, 1\rangle$ in $\mathcal{G}$, where Player 1 has the advantage, and $i \in \{0, \ldots, B_1\}$. If $M_c(i-1, i) = M_c(i, i-1) = 2$, then $M_c(i, i) = 2$, and if $M_c(i, i) = 2$, then $M_c(i+1, i) = 2$. Consider a configuration $c = \langle v, B_1, B_2, 2\rangle$ in $\mathcal{G}$, where Player 2 has the advantage, and $i \in \{0, \ldots, B_2\}$. If $M_c(i-1, i) = M_c(i, i-1) = 1$, then $M_c(i, i) = 1$, and if $M_c(i, i-1) = 2$, then $M_c(i, i) = 2$.*

In the full version, we combine the lemmas above to show that advantage-based tie-breaking gives rise to local determinacy and thus obtain the following theorem.

▶ **Theorem 22.** *Müller bidding games with advantage-based tie-breaking are determined.*

We turn to study computational complexity of bidding games. Let $\text{BID}_{\alpha,\text{adv}}$ be the class of bidding games with advantage-based tie-breaking and objective α, and let $\text{BID-WIN}_{\alpha,\text{adv}}$ be the respective decision problem. Recall that TB-WIN_α is the decision problem for turn-based games. The upper bound in the following theorem is implied from determinacy and the lower bound is similar to Theorem 15 and can be found in the full version.

▶ **Theorem 23.** *For a qualitative objective α, the complexity of $\textit{TB-WIN}_\alpha$ and $\textit{BID-WIN}_{\alpha,adv}$ coincide.*

7 Strongly-Connected Games

Reasoning about strongly-connected games is key to the solution in continuous-bidding infinite-duration games [7, 8, 10]. It is shown that in a strongly-connected continuous-bidding game, with every initial positive budget, a player can force the game to visit every vertex

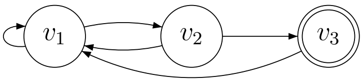


Figure 8 A strongly-connected Büchi game in which Player 1 loses with every initial budget.

infinitely often. It follows that in a strongly-connected Büchi game $\mathcal{G}$ with at least one accepting state, Player 1 wins with every positive initial budget. We show a similar result in discrete-bidding games in two cases, where the proof can be found in the full version.

▶ **Theorem 24.** *Consider a strongly-connected bidding game $\mathcal{G}$ in which tie-breaking is either resolved randomly or by a transducer that always prefers Player 1. Then, for every pair of initial budgets, Player 1 can force visiting every vertex in $\mathcal{G}$ infinitely often with probability 1.*

In [20], it is roughly stated that, with advantage-based tie-breaking, as the budgets tend to infinity, the game "behaves" similarly to a continuous-bidding game. We show that infinite-duration discrete-bidding games can be quite different from their continuous counterparts; namely, we show a Büchi game $\mathcal{G}$ such that under continuous-bidding, Player 1 wins in $\mathcal{G}$ with every pair of initial budgets, and under discrete-bidding, Player 1 loses in $\mathcal{G}$ with every pair of initial budgets.

▶ **Theorem 25.** *There is a strongly-connected Büchi discrete-bidding game with advantage-based tie-breaking such that Player 1 loses with every pair of initial budgets.*

Proof. Suppose the game that is depicted in Fig. 8 starts at vertex v_1 with initial budgets $B_1 \in \mathbb{N}$ and $B_2 = 0$. Player 2 always bids 0, uses the advantage when he has it, and, upon winning, stays in v_1 and moves from v_2 to v_1. Note that in order to visit v_3, Player 1 needs to win two biddings in a row; in v_1 and v_2. Thus, in order to visit v_3, he must "invest" a unit of budget, meaning that the number of visits to v_3 is bounded by B_1. ◀

8 Discussion and Future Work

We study discrete-bidding infinite-duration bidding games and identify large fragments of bidding games that are determined. Bidding games are a subclass of concurrent games. We are not aware of other subclasses of concurrent games that admit determinacy. We find it an interesting future direction to extend the determinacy we show here beyond bidding games. Weaker versions of determinacy in fragments of concurrent games have been previously studied [37].

We focused on bidding games with "Richman" bidding and it is interesting to study other bidding games with other bidding rules. Discrete-bidding has previously been studied in combination with *all-pay* bidding [30] in which both players pay their bid to the other player. In addition, it is interesting to study discrete-bidding games with quantitative objectives and non-zero-sum games, which were previously studied only for continuous bidding [7, 8, 29].

This work belongs to a line of works that transfer concepts and ideas between the areas of formal verification and algorithmic game theory [33]. Examples of works in the intersection of the two fields include logics for specifying multi-agent systems [2, 17, 31], studies of equilibria in games related to synthesis and repair problems [16, 14, 22, 1], non-zero-sum games in formal verification [18, 13], and applying concepts from formal methods to *resource allocation games* such as rich specifications [11], efficient reasoning about very large games [6, 24], and a dynamic selection of resources [9].

References

1 S. Almagor, G. Avni, and O. Kupferman. Repairing Multi-Player Games. In *Proc. 26th CONCUR*, pages 325–339, 2015.

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

3 B. Aminof and S. Rubin. First-cycle games. *Inf. Comput.*, 254:195–216, 2017.

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

5 N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on Ethereum smart contracts. *IACR Cryptology ePrint Archive*, 2016:1007, 2016.

6 G. Avni, S. Guha, and O. Kupferman. An Abstraction-Refinement Methodology for Reasoning about Network Games. *Games*, 9(3), 2018.

7 G. Avni, T. A. Henzinger, and V. Chonev. Infinite-Duration Bidding Games. In *Proc. 28th CONCUR*, volume 85 of *LIPIcs*, pages 21:1–21:18, 2017.

8 G. Avni, T. A. Henzinger, and R. Ibsen-Jensen. Infinite-Duration Poorman-Bidding Games. In *Proc. 14th WINE*, volume 11316 of *LNCS*, pages 21–36. Springer, 2018.

9 G. Avni, T. A. Henzinger, and O. Kupferman. Dynamic Resource Allocation Games. In *Proc. 9th SAGT*, pages 153–166, 2016.

10 G. Avni, T. A. Henzinger, and Đ. Žikelić. Bidding Mechanisms in Graph Games. In *In Proc. 44th MFCS*, 2019.

11 G. Avni, O. Kupferman, and T. Tamir. Network-formation games with regular objectives. *Inf. Comput.*, 251:165–178, 2016.

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

13 T. Brihaye, V. Bruyère, J. De Pril, and H. Gimbert. On Subgame Perfection in Quantitative Reachability Games. *Logical Methods in Computer Science*, 9(1), 2012.

14 K. Chatterjee. Nash Equilibrium for Upward-Closed Objectives. In *Proc. 15th CSL*, volume 4207, pages 271–286. Springer, 2006.

15 K. Chatterjee, A. K. Goharshady, and Y. Velner. Quantitative Analysis of Smart Contracts. In *Proc. 27th ESOP*, pages 739–767, 2018.

16 K. Chatterjee, T. A. Henzinger, and M. Jurdzinski. Games with secure equilibria. *Theor. Comput. Sci.*, 365(1-2):67–82, 2006.

17 K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. *Inf. Comput.*, 208(6):677–693, 2010.

18 K. Chatterjee, R. Majumdar, and M. Jurdzinski. On Nash Equilibria in Stochastic Games. In *Proc. 13th CSL*, pages 26–40, 2004.

19 A. Condon. On Algorithms for Simple Stochastic Games. In *Proc. DIMACS*, pages 51–72, 1990.

20 M. Develin and S. Payne. Discrete Bidding Games. *The Electronic Journal of Combinatorics*, 17(1):R85, 2010.

21 E.A. Emerson and C. Jutla. Tree Automata, μ-Calculus and Determinacy. In *Proc. 32nd FOCS*, pages 368–377, 1991.

22 D. Fisman, O. Kupferman, and Y. Lustig. Rational Synthesis. In *Proc. 16th TACAS*, pages 190–204, 2010.

23 E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500. Springer, 2002.

24 O. Kupferman and T. Tamir. Hierarchical Network Formation Games. In *Proc. 23rd TACAS*, pages 229–246, 2017.

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

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

27 D. A. Martin. The Determinacy of Blackwell Games. *J. Symb. Log.*, 63(4):1565–1581, 1998.

28 D.A. Martin. Borel Determinacy. *Annals of Mathematics*, 65:363–371, 1975.

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

30 M. Menz, J. Wang, and J. Xie. Discrete All-Pay Bidding Games. *CoRR*, abs/1504.02799, 2015. arXiv:1504.02799.

31 F. Mogavero, A. Murano, G. Perelli, and M. Y. Vardi. Reasoning About Strategies: On the Model-Checking Problem. *ACM Trans. Comput. Log.*, 15(4):34:1–34:47, 2014.

32 S. Muthukrishnan. Ad Exchanges: Research Issues. In *Proc. 5th WINE*, pages 1–12, 2009.

33 N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, 2007.

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

35 A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *Proc. 16th POPL*, pages 179–190, 1989.

36 M.O. Rabin. Decidability of Second Order Theories and Automata on Infinite Trees. *Transaction of the AMS*, 141:1–35, 1969.

37 S. Le Roux. Concurrent Games and Semi-Random Determinacy. In *Proc. 43rd MFCS*, pages 40:1–40:15, 2018.

Energy Mean-Payoff Games

Véronique Bruyère
Université de Mons (UMONS), Belgium
veronique.bruyere@umons.ac.be

Quentin Hautem[1]
Université de Mons (UMONS), Belgium
q.hautem@gmail.com

Mickael Randour[2]
Université de Mons (F.R.S.-FNRS & UMONS), Belgium
mickael.randour@gmail.com

Jean-François Raskin
Université libre de Bruxelles (ULB), Belgium
jraskin@ulb.ac.be

Abstract

In this paper, we study one-player and two-player energy mean-payoff games. Energy mean-payoff games are games of infinite duration played on a finite graph with edges labeled by 2-dimensional weight vectors. The objective of the first player (the protagonist) is to satisfy an energy objective on the first dimension and a mean-payoff objective on the second dimension. We show that optimal strategies for the first player may require infinite memory while optimal strategies for the second player (the antagonist) do not require memory. In the one-player case (where only the first player has choices), the problem of deciding who is the winner can be solved in polynomial time while for the two-player case we show co-NP membership and we give effective constructions for the infinite-memory optimal strategies of the protagonist.

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 two-player zero-sum games played on graphs, energy and mean-payoff objectives, complexity study and construction of optimal strategies

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.21

Related Version A full version of the paper is available at [14], `https://arxiv.org/pdf/1907.01359.pdf`.

Funding Work partially supported by the PDR project *Subgame perfection in graph games* (F.R.S.-FNRS), the grant n°F.4520.18 *ManySynth* (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 & FWO), and the COST Action 16228 *GAMENET* (European Cooperation in Science and Technology).

1 Introduction

Graph games with ω-regular objectives are a canonical mathematical model to formalize and solve the reactive synthesis problem [33]. Extensions of graph games with quantitative objectives have been considered more recently as a model where, not only the correctness,

[1] supported by a FRIA fellowship
[2] F.R.S.-FNRS Research Associate

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 21; pp. 21:1–21:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

but also the quality of solutions for the reactive synthesis problem can be formalized and optimized. A large effort has been invested in studying games with various kinds of objectives, see e.g. [5, 12, 15, 19, 21, 22, 25, 35, 36], see also Chapter 27 of [3] and the survey [13].

Two particularly important classes of objectives are *mean-payoff* and *energy* objectives. In a mean-payoff game, the edges of the game graph are labeled with integer weights that model payoffs received by the first player (the protagonist) and paid by the second player (the antagonist) when the edge is taken. The game is played for infinitely many rounds, and the protagonist aims at maximizing the mean value of edges traversed during the game while the antagonist tries to minimize this mean value. Mean-payoff games have been studied in [25] where it is shown that memoryless optimal strategies exist for both players. As a corollary of this result, mean-payoff games can be decided in NP ∩ co-NP. While pseudo-polynomial time algorithms for solving mean-payoff games have been developed in [12, 36] as well as the recent pseudo-quasi-polynomial time algorithm in [24], it is a long standing open question whether or not those games can be solved in polynomial time. Energy games were defined more recently in [16]. In an energy game, edges are also labeled with integer weights that represent gains or losses of energy. In such a game, the protagonist tries to build an infinite path for which the total sum of energy in all the prefixes is bounded from below, while the antagonist has the opposite goal. Energy games can also be decided in NP ∩ co-NP and it is known that they are *inter-reducible* with mean-payoff games [5].

Energy mean-payoff games that combine an energy and a mean-payoff objectives have not been yet studied. This is the main goal of this paper. It is a challenging problem for several reasons. First, multi-dimensional *homogeneous* extensions of mean-payoff and energy games have been studied in a series of recent contributions [21, 29, 34, 35], and those works show that when going from one dimension to several, the close relationship between mean-payoff games and energy games is lost and specific new techniques need to be designed for solving those extensions. Second, *pushdown mean-payoff games* have been studied in [22] and shown to be undecidable. Decision problems for energy mean-payoff games can be reduced to decision problems of pushdown mean-payoff games, even to the subclass of pushdown mean-payoff games with a one-letter stack alphabet. Unfortunately, pushdown mean-payoff games are undecidable in general and to the best of our knowledge the one-letter stack alphabet case has not been studied.

Main contributions. In this paper, we prove that energy mean-payoff games are decidable. More precisely, their decision problems lie in co-NP (Theorem 7) for both cases of strict and non-strict inequality in the threshold constraint for the mean-payoff objective. To obtain this result, we first study *one-player* energy mean-payoff games and characterize precisely the game graphs in which $\mathcal{P}_1$ (the protagonist) can build an infinite path that satisfies the energy mean-payoff objective (Theorem 5 and Theorem 6). This characterization leads to polynomial time algorithms to solve the decision problems in the one-player case (Theorem 3). Then we show that in *two-player* energy mean-payoff games memoryless optimal strategies always exist for $\mathcal{P}_2$ (the antagonist) who aims at spoiling the energy mean-payoff objective of $\mathcal{P}_1$ (Proposition 8). Combined with the polynomial time algorithms for the one-player case, this result leads to co-NP membership of the decision problems. While the memoryless result for $\mathcal{P}_2$ allows us to understand how this player should play in energy mean-payoff games, it does not prescribe how $\mathcal{P}_1$ should play from winning vertices. To show how to effectively construct optimal strategies for $\mathcal{P}_1$, we consider a reduction to *4-dimensional energy games* in case of strict inequality for mean-payoff objective (Proposition 12). With the result of [29], this implies the existence of finite-memory strategies for $\mathcal{P}_1$ to play optimally and of a pseudo-polynomial time algorithm to solve those instances. For non-strict inequalities, this reduction cannot be applied as, even for the one-player case, infinite-memory strategies are sometimes necessary to play optimally. In this case, we show how we can combine an infinite number of finite-memory strategies, that are played in sequence, in order to play optimally (Proposition 13).

Related work. As already mentioned, multi-dimensional conjunctive extensions of mean-payoff games and multi-dimensional conjunctive extensions of energy games have been considered [18, 21, 35]. Deciding the existence of a winning strategy for $\mathcal{P}_1$ in those games is co-NP-complete. Games with any Boolean combination of mean-payoff objectives have been shown undecidable in [34]. Games with mean-payoff objectives and ω-regular constraints have been studied in [20], while games with energy objectives and ω-regular constraints have been studied in [17], and their multi-dimensional extensions in [2, 21, 23].

In [29], the authors have studied multi-dimensional energy games for the fixed initial credit and provided a pseudo-polynomial time algorithm to solve them when the number of dimensions is fixed. Energy games with bounds on the energy level have been studied in [26, 28]. Games with the combination of an energy objective and an average-energy objective are investigated in [6, 7]. This seemingly related class of games is actually quite different from the energy mean-payoff games studied in this paper: e.g., they are EXPSPACE-hard whereas our games are in co-NP. Infinite-state energy games are investigated in [1] where energy objectives are studied on infinite game structures, induced by one-counter automata or pushdown automata. Some work on other models dealing with energy have been studied, as battery edge systems [4] and consumption games [8]. In the latter games, minimization of running costs have also been investigated [10]. Optimizing the expected mean-payoff in energy MDP's have been studied in [11]. In [32], Kucera presents an overview of results related to games and counter automata, which are close to energy constraints.

We now discuss *mean-payoff pushdown games* [22] in more details. In those games, a stack is associated with a finite game structure, and players move from vertex to vertex while applying operations on the stack. Those operations are *push* a letter, *pop* a letter or *skip* and can be respectively represented with weights 1, -1 and 0. The authors show that one-player pushdown games can be solved in polynomial time, thanks to the existence of *pumpable paths*. Moreover, already in this case, $\mathcal{P}_1$ needs infinite memory to win in mean-payoff pushdown games. In the two-player setting, determining the winner is undecidable. Doing a straight reduction of one-player energy mean-payoff games to one-player mean-payoff pushdown games would lead to a pseudo-polynomial solution, whereas we show here that we can solve the former games in polynomial time. In addition, we cannot use the concept of pumpable paths to obtain those results as the construction of [22] is inherent to the behavior of the stack of mean-payoff pushdown games. Indeed, after one step, the height of the stack can only change of one unity $(+1, -1, 0)$, whereas in energy mean-payoff games, the energy level can vary from $-W$ to $+W$, for an arbitrarily large integer $W \in \mathbb{N}$.

Structure of the paper. In Sect. 2, we introduce the necessary notations and preliminaries to this work. In Sect. 3, we study the one-player energy mean-payoff games. In Sect. 4, we study the two-player energy mean-payoff games.

2 Preliminaries

In this section, we introduce energy mean-payoff games and the related decision problems studied in this paper.

Games structures. A *game structure* is a weighted directed graph $G = (V, V_1, V_2, E, w)$ such that V_1, V_2 form a partition of the finite set V, V_i is the set of vertices controlled by player $\mathcal{P}_i$, $i \in \{1,2\}$, $E \subseteq V \times V$ is the set of edges such that for all $v \in V$, there exists $v' \in V$ such that $(v,v') \in E$, and $w = (w_1, w_2) : E \to \mathbb{Z}^2$ is a weight function that assigns a pair of weights $w(e) = (w_1(e), w_2(e))$ to each edge $e \in E$. In the whole paper, we denote by $|V|$ the number of vertices of V, by $|E|$ the number of edges of E, and by $||E|| \in \mathbb{N}_0$ the largest absolute value used by the weight function w. We say that a game structure is a *player-i game structure* when player $\mathcal{P}_i$ controls all the vertices, that is, $V_i = V$.

A *play* in G from an *initial vertex* v_0 is an infinite sequence $\rho = \rho_0\rho_1 \dots \rho_k \dots$ of vertices such that $\rho_0 = v_0$ and $(\rho_k, \rho_{k+1}) \in E$ for all $k \geq 0$. A *factor* of ρ, denoted by $\rho[k,\ell]$, is the finite sequence $\rho_k\rho_{k+1} \dots \rho_\ell$. When $k = 0$, we say that $\rho[0,\ell]$ is the *prefix* of length ℓ of ρ. The *suffix* $\rho_k\rho_{k+1} \dots$ of ρ is denoted by $\rho[k,\infty]$. The set of plays in G is denoted by $\mathsf{Plays}(G)$ or simply Plays. A path or a cycle is *simple* if there are no two occurrences of the same vertex (except for the first and last vertices in the cycle). A *multicycle* $\mathcal{C}$ is a multiset of simple cycles (that may or may not be connected to each other). We extend the weight function w to paths (resp. cycles, multicycles) as the sum $w(\pi) = (w_1(\pi), w_2(\pi))$ of the weights of their edges. In particular, for a multicycle $\mathcal{C}$, we have $w(\mathcal{C}) = \sum_{\pi \in \mathcal{C}} w(\pi)$.

Given a path $\pi = \pi_0\pi_1 \cdots \pi_n$, we consider its *cycle decomposition* into a *multiset of simple cycles* as follows. We push successively vertices $\pi_0, \pi_1, \dots$ onto a stack. Whenever we push a vertex π_ℓ equal to a vertex π_k already in the stack, i.e. a simple cycle $C = \pi_k \cdots \pi_\ell$ is formed, we remove this cycle from the stack except π_k (we remove all the vertices until reaching π_k that we let in the stack) and add C to the cycle decomposition multiset of π. The cycle decomposition of a play $\rho = \rho_0\rho_1 \dots$ is defined similarly.

For each dimension $j \in \{1,2\}$, the weight or *energy level* of the prefix $\rho[0,k]$ of a play ρ is $w_j(\rho[0,k])$, and the *mean-payoff-inf* (resp. *mean-payoff-sup*) of ρ is $\underline{\mathsf{MP}}_j(\rho) = \liminf_{k\to\infty} \frac{1}{k} \cdot w_j(\rho[0,k])$ (resp. $\overline{\mathsf{MP}}_j(\rho) = \limsup_{k\to\infty} \frac{1}{k} \cdot w_j(\rho[0,k])$). The following properties hold for both mean-payoff values. First, they are prefix-independent, that is, $\underline{\mathsf{MP}}_j(\pi\rho) = \underline{\mathsf{MP}}_j(\rho)$ and $\overline{\mathsf{MP}}_j(\pi\rho) = \overline{\mathsf{MP}}_j(\rho)$ for all finite paths π. Second for a play $\rho = \rho_0 \dots \rho_{k-1}(\rho_k \dots \rho_l)^\omega$ that is eventually periodic, its mean-payoff-inf and mean-payoff-sup values coincide and are both equal to the average weight of the cycle $\rho_k \dots \rho_l\rho_k$, that is, $\frac{1}{l-k+1} \cdot w_j(\rho_k \dots \rho_l\rho_k)$.

Strategies. Given a game structure G, a *strategy* σ_i for player $\mathcal{P}_i$ is a function $V^* \cdot V_i \to V$ that assigns to each path πv ending in a vertex $v \in V_i$ a vertex v' such that $(v,v') \in E$. Such a strategy σ_i is *memoryless* if it only depends on the last vertex of the path, i.e. $\sigma_i(\pi v) = \sigma_i(\pi' v)$ for all $\pi v, \pi' v \in V^* \cdot V_i$. It is a *finite-memory* strategy if it can be encoded by a deterministic *Moore machine* $\mathcal{M} = (M, m_0, \alpha_U, \alpha_N)$ where M is a finite set of states (the memory of the strategy), $m_0 \in M$ is an initial memory state, $\alpha_U : M \times V \to M$ is an update function, and $\alpha_N : M \times V_i \to V$ is a next-move function. Such a machine defines a strategy σ_i such that $\sigma_i(\pi v) = \alpha_N(\widehat{\alpha}_U(m_0, \pi), v)$ for all paths $\pi v \in V^* \cdot V_i$, where $\widehat{\alpha}_U$ extends α_U to paths as expected. The *memory size* of σ_i is then the size $|M|$ of $\mathcal{M}$. In particular σ_i is memoryless when it has memory size one.

Given a strategy σ_i for $\mathcal{P}_i$, a play ρ is *consistent* with σ_i if for all its prefixes $\rho[0,k] \in V^* \cdot V_i$, we have $\rho_{k+1} = \sigma_i(\rho[0,k])$. A finite path π consistent with σ_i is defined similarly. Given a finite-memory strategy σ_i and its Moore machine $\mathcal{M}$, we denote by $G(\sigma_i)$ the game structure obtained as the product of G with $\mathcal{M}$. Notice that the set of plays from an initial vertex v_0 that are consistent with σ_i is then exactly the set of plays in $G(\sigma_i)$ starting from (v_0, m_0) where m_0 is the initial memory state of $\mathcal{M}$.

Objectives. Given a game structure G and an initial vertex v_0, an *objective* for player $\mathcal{P}_1$ is a set of plays $\Omega \subseteq \mathsf{Plays}(G)$. Given a strategy σ_1 for $\mathcal{P}_1$, we say that σ_1 is *winning for* $\mathcal{P}_1$ *from* v_0 if all plays $\rho \in \mathsf{Plays}(G)$ from v_0 that are consistent with σ_1 satisfy $\rho \in \Omega$. Given a strategy σ_2 for $\mathcal{P}_2$, we say that σ_2 is *winning for* $\mathcal{P}_2$ *from* v_0 if all plays $\rho \in \mathsf{Plays}(G)$ from v_0 that are consistent with σ_2 satisfy $\rho \notin \Omega$.

We here consider the following objectives for dimension $j \in \{1, 2\}$:

- *Energy objective.* Given an *initial credit* $c_0 \in \mathbb{N}$, the objective $\mathsf{Energy}_j(c_0) = \{\rho \in \mathsf{Plays}(G) \mid \forall k \geq 0, c_0 + w_j(\rho[0,k]) \geq 0\}$ requires that the energy level remains always nonnegative in dimension j.
- *Mean-payoff-inf objective.* The objective $\underline{\mathsf{MP}}_j(\sim 0) = \{\rho \in \mathsf{Plays}(G) \mid \underline{\mathsf{MP}}_j(\rho) \sim 0\}$ with $\sim \in \{>, \geq\}$ requires that the mean-payoff-inf value is ~ 0 in dimension j.
- *Mean-payoff-sup objective.* The objective $\overline{\mathsf{MP}}_j(\sim 0) = \{\rho \in \mathsf{Plays}(G) \mid \overline{\mathsf{MP}}_j(\rho) \sim 0\}$ with $\sim \in \{>, \geq\}$ requires that the mean-payoff-sup value is ~ 0 in dimension j.

Notice that it is not a restriction to work with threshold 0 in mean-payoff-inf/sup objectives. Indeed arbitrary thresholds $\frac{a}{b} \in \mathbb{Q}$ can be reduced to threshold 0 by replacing the weight function w of G by the function $b \cdot w - a$.

Decision problems. In this paper we consider the following four variants of a decision problem implying an energy objective on the first dimension and a mean-payoff objective on the second dimension. Let $\sim \in \{>, \geq\}$:

- The *energy mean-payoff decision problem* $\mathsf{E} \cap \underline{\mathsf{MP}}^{\sim 0}$ asks, given a game structure G and an initial vertex v_0, to decide whether there exist an initial credit $c_0 \in \mathbb{N}$ and a winning strategy σ_1 for player $\mathcal{P}_1$ from v_0 for the objective $\Omega = \mathsf{Energy}_1(c_0) \cap \underline{\mathsf{MP}}_2(\sim 0)$.
- The *energy mean-payoff decision problem* $\mathsf{E} \cap \overline{\mathsf{MP}}^{\sim 0}$ asks, given a game structure G and an initial vertex v_0, to decide whether there exist an initial credit $c_0 \in \mathbb{N}$ and a winning strategy σ_1 for player $\mathcal{P}_1$ from v_0 for the objective $\Omega = \mathsf{Energy}_1(c_0) \cap \overline{\mathsf{MP}}_2(\sim 0)$.

In this context, we also use the terminology of *energy mean-payoff objectives* or *energy mean-payoff games*. Let us give two illustrating examples.

▶ **Example 1.** Consider the player-1 game structure G depicted in Figure 1. Consider the cycle $C = v_0 v_0 v_0 v_1 v_1 v_1 v_0$ that loops twice on v_0, goes to v_1, loops twice on v_1, and comes back to v_0. Observe that $w(C) = (w_1(C), w_2(C)) = (0, 2)$. Hence $\mathcal{P}_1$ has a winning strategy, that consists in looping forever in this cycle C, for all four variants of the energy mean-payoff decision problem.

▶ **Example 2.** Consider now the player-1 game structure G depicted in Figure 2. It differs from the previous game structure only by the weight $(-1, 1)$ (instead of $(-1, 3)$) of the edge (v_1, v_1). We claim that $\mathcal{P}_1$ has no winning strategy for any of the two decision problems $\mathsf{E} \cap \underline{\mathsf{MP}}^{>0}$ and $\mathsf{E} \cap \overline{\mathsf{MP}}^{>0}$. We will explain why in Section 3. However $\mathcal{P}_1$ has a winning strategy for both problems $\mathsf{E} \cap \underline{\mathsf{MP}}^{\geq 0}$ and $\mathsf{E} \cap \overline{\mathsf{MP}}^{\geq 0}$ with initial credit $c_0 = 0$. Such a strategy consists in repeatedly executing the following round, with $Z = 1$ initially, and Z incremented by 1 after each round: loop Z times on v_0, go to v_1, loop Z times on v_1, and come back to v_0. Such a strategy with infinite memory is clearly winning for the energy objective. It is also winning for the mean-payoff-inf objective because with Z increased by 1 at each round, the cost of moving from v_0 to v_1 and from v_1 to v_0 becomes negligible.

No finite-memory strategy is winning in this case. Indeed assume the contrary: there exists a winning strategy that induces a cycle C in which it loops forever. This cycle necessarily uses both simple cycles $C_0 = (v_0, v_0)$ and $C_1 = (v_1, v_1)$ as the strategy is winning. As these cycles are not connected, C has to also use the simple cycle $C_3 = (v_0, v_1, v_0)$. As $w(C_1) = -w(C_2)$ and $w(C_3) = (0, -2)$, it is easy to see that it is impossible that this strategy satisfies both energy and mean-payoff-inf/sup objectives simultaneously.

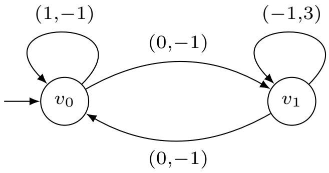


Figure 1 Energy mean-payoff game where $\mathcal{P}_1$ wins with finite memory for problems $\mathsf{E} \cap \underline{\mathsf{MP}}^{>0}$ and $\mathsf{E} \cap \overline{\mathsf{MP}}^{>0}$.

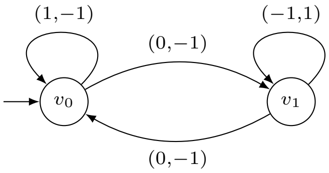


Figure 2 Energy mean-payoff game where $\mathcal{P}_1$ needs infinite memory to win for problems $\mathsf{E} \cap \underline{\mathsf{MP}}^{\geq 0}$ and $\mathsf{E} \cap \overline{\mathsf{MP}}^{\geq 0}$.

3 One-player setting

Within this section, we investigate player-1 game structures, that is, game structures where player $\mathcal{P}_1$ is the only one to play. In this context, $\mathcal{P}_1$ has a winning strategy for the energy mean-payoff objective for some initial credit c_0 if and only if there exists a play belonging to this objective. For player-1 game structures, we show that the energy mean-payoff decision problem can be solved in polynomial time for all of its four variants. However depending on the used relation $\sim \in \{>, \geq\}$ for the mean-payoff objective, memory requirements for winning strategies of $\mathcal{P}_1$ differ. We already know that $\mathcal{P}_1$ needs infinite memory in case of non-strict inequalities by Example 2. In case of strict inequalities, we show that finite-memory strategies are always sufficient for $\mathcal{P}_1$, as in Example 1. All these results will be useful in Section 4 when we will investigate the general case of two-player energy mean-payoff games.

▶ **Theorem 3.** *The energy mean-payoff decision problem for player-1 game structures can be solved in polynomial time. Moreover,*

- *for both problems $\mathsf{E} \cap \underline{\mathsf{MP}}^{>0}$ and $\mathsf{E} \cap \overline{\mathsf{MP}}^{>0}$, pseudo-polynomial-memory strategies are sufficient and necessary for $\mathcal{P}_1$ to win;*
- *for both problems $\mathsf{E} \cap \underline{\mathsf{MP}}^{\geq 0}$ and $\mathsf{E} \cap \overline{\mathsf{MP}}^{\geq 0}$, in general, $\mathcal{P}_1$ needs infinite memory to win.*

To prove Theorem 3, we characterize the existence of a winning strategy for $\mathcal{P}_1$ for some initial credit c_0 by the existence of a particular cycle or multicycle, that we call *good*.

▶ **Definition 4.** *Let G be a game structure and v_0 be an initial vertex.*

- *We say that a cycle C is a* good cycle *if $w_1(C) \geq 0$ and $w_2(C) > 0$. A good cycle C is* reachable *if it is reachable from v_0.*
- *We say that a multicycle $\mathcal{C}$ is a* good multicycle *if $w_1(\mathcal{C}) \geq 0$ and $w_2(\mathcal{C}) \geq 0$. A good multicyle $\mathcal{C}$ is* reachable *if all its simple cycles are in the same connected component reachable from v_0.*

There exists a simple characterization of the existence of a winning strategy for $\mathcal{P}_1$ for either the objective $\mathsf{Energy}_1(c_0) \cap \underline{\mathsf{MP}}_2(> 0)$ or the objective $\mathsf{Energy}_1(c_0) \cap \overline{\mathsf{MP}}_2(> 0)$ for some initial credit c_0: both are equivalent to the existence of a reachable good cycle.

▶ **Theorem 5.** *Let G be a player-1 game structure and v_0 be an initial vertex. The following assertions are equivalent.*

1. *There exist an initial credit c_0 and a winning strategy for $\mathcal{P}_1$ from v_0 for the objective* $\mathsf{Energy}_1(c_0) \cap \underline{\mathsf{MP}}_2(> 0)$.
2. *There exist an initial credit c_0 and a winning strategy for $\mathcal{P}_1$ from v_0 for the objective* $\mathsf{Energy}_1(c_0) \cap \overline{\mathsf{MP}}_2(> 0)$.
3. *There exists a reachable good cycle.*

In case of non-strict inequalities, there exists also a simple characterization: $\mathcal{P}_1$ can win for either the objective $\mathsf{Energy}_1(c_0) \cap \underline{\mathsf{MP}}_2(\geq 0)$ or the objective $\mathsf{Energy}_1(c_0) \cap \overline{\mathsf{MP}}_2(\geq 0)$ for some initial credit c_0 if and only if there exists a reachable good multicycle.

▶ **Theorem 6.** *Let G be a player-1 game structure and v_0 be an initial vertex. The following assertions are equivalent.*

1. *There exist an initial credit c_0 and a winning strategy for $\mathcal{P}_1$ from v_0 for the objective* $\mathsf{Energy}_1(c_0) \cap \underline{\mathsf{MP}}_2(\geq 0)$.
2. *There exist an initial credit c_0 and a winning strategy for $\mathcal{P}_1$ from v_0 for the objective* $\mathsf{Energy}_1(c_0) \cap \overline{\mathsf{MP}}_2(\geq 0)$.
3. *There exists a reachable good multicycle.*

A similar characterization appears for multi-mean-payoff games and multi-energy games studied in [35]: when the objective is an intersection of several mean-payoff-inf objectives (resp. several energy objectives), and when he plays alone, $\mathcal{P}_1$ has a winning strategy if and only if there exists a reachable non negative multicycle (resp. a reachable non negative cycle) in the game structure. Nevertheless, the proofs of those results differ substantially from the proofs of our results.

Due to the lack of space, we only give the main ideas of the proofs (see [14] for more details):

1. We begin by illustrating the statements of Theorems 5 and 6 with the two previous examples. Let us first come back to the game structure of Figure 1. The cycle C mentioned in Example 1 is a reachable good cycle since $w(C) = (0, 2)$. By Theorem 5, it follows that $\mathcal{P}_1$ is winning for the energy mean-payoff decision problem with strict inequalities (and thus also with non-strict inequalities), as already observed in Example 1. Let us now come back to the game structure of Figure 2. Recall from Example 2 that there exists a winning strategy for $\mathcal{P}_1$ in case of non-strict inequalities, but no winning strategy in case of strict inequalities. By Theorem 6, there should exist a reachable good multicycle. Indeed, consider the multicycle $\mathcal{C} = \{C, C'\}$ with $C = (v_0, v_0)$ and $C' = (v_1, v_1)$: we have $w(\mathcal{C}) = w(C) + w(C') = (1, -1) + (-1, 1) = (0, 0)$. Moreover by Theorem 5, there is no reachable good cycle in this game.
2. By Theorems 5 and 6, solving the energy mean-payoff decision problem reduces to decide whether there exists a reachable good cycle (resp. multicycle). This can be tested in polynomial time thanks to a variant of a result in [31] that states that deciding the existence of a cycle (resp. multicycle) of weight $(0, 0)$ can be done in polynomial time. This established the polynomial complexity stated in Theorem 3.
3. Theorem 5 is proved as follows. For Implication (3) ⇒ (1), a winning strategy for $\mathcal{P}_1$ consists in reaching the good cycle and looping in it forever. Implication (1) ⇒ (2) is trivial since $\overline{\mathsf{MP}}_2(\rho) \geq \underline{\mathsf{MP}}_2(\rho)$ for all plays ρ. However, the proof of Implication (2) ⇒ (3) is rather technical, it is thus detailed in [14].
4. The proof of Theorem 6 requires a more precise characterization by good cycles and multicycles. We show that the existence of a good cycle is equivalent to the existence of

- either one good cycle that is simple,
- or two simple cycles C, C' with respective weight vectors $w(C) = (-x, y)$ and $w(C') = (x', -y')$ that satisfy $x, x', y \in \mathbb{N}_0$ and $y' \in \mathbb{N}$ and make an angle $< 180^o$ (see Figure 3).

We show a similar equivalence for the existence of a good multicycle with the existence of

- either a good multicycle $\mathcal{C} = \{C\}$ composed of a unique simple cycle C,
- or two simple cycles C, C' as before with the difference that $w(C), w(C')$ make an angle $\leq 180^o$ (instead of $< 180^o$).

The proof of these results is based on the cycle decomposition of paths and on geometrical arguments on the weights of those simple cycles.

Let us illustrate this characterization with our two running examples. In case of Figure 1, the good cycle is characterized by the two cycles $C = (v_1, v_1)$, $C' = (v_0, v_0)$ with respective weights $(-1, 3)$, $(1, -1)$. In case of Figure 2, the good multicycle is characterized by the two cycles $C = (v_1, v_1)$, $C' = (v_0, v_0)$ with respective weights $(-1, 1)$, $(1, -1)$. Moreover one can check that there is no good cycle (the conditions given in the characterization do not hold in Figure 2).

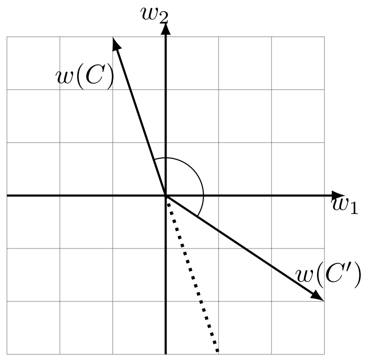


Figure 3 Geometrical view of the characterization for good cycles.

5. With the characterization given previously in Item 4., in case of strict inequalities, pseudo-polynomial-memory strategies are sufficient for $\mathcal{P}_1$ to win, as stated in Theorem 3. Indeed when there exist two simple cycles C, C' with weight vectors $w(C)$, $w(C')$ as in Figure 3, one can construct a good cycle as follows. There always exists a linear combination of vectors $w(C)$, $w(C')$, with pseudo-polynomial positive coefficients, that is $> (0, 0)$ and that balances the cost of moving from C to C' and from C' to C (this is however not possible when those vectors make an angle of 180^o). The fact that in case of strict inequalities, pseudo-polynomial-memory strategies are necessary for $\mathcal{P}_1$ to win is proved in [14]. Notice that in case of non-strict inequalities, Theorem 3 states that $\mathcal{P}_1$ needs infinite memory to win, that we already know from Example 2.
6. Theorem 6 is proved as follows. Implication (1) $\Rightarrow$ (2) is immediate. For Implication (3) $\Rightarrow$ (1), with the two simple cycles C, C' of the characterization given in Item 4., one can construct a winning strategy with infinite memory for $\mathcal{P}_1$ that is similar to the strategy of Example 2. To prove (2) $\Rightarrow$ (3), suppose that $\mathcal{P}_1$ is winning for the objective $\mathsf{Energy}_1(c_0) \cap \overline{\mathsf{MP}}_2(\geq 0)$ for some c_0. Then he is also winning for $\mathsf{Energy}_1(c_0) \cap \overline{\mathsf{MP}}_2(> -\epsilon)$ for all $\epsilon > 0$. We consider the game structure G_ϵ obtained from G by replacing function w_2 by function $w_2 + \epsilon$. Hence $\mathcal{P}_1$ is now winning in this game G_ϵ for the objective $\mathsf{Energy}_1(c_0) \cap \overline{\mathsf{MP}}_2(> 0)$, and by Theorem 5 there exists a reachable good cycle in G_ϵ. Therefore, with the characterization of Item 4., for each ϵ, there exist either one good

simple cycle C_ϵ, or two simple cycles C_ϵ, C'_ϵ with weight vectors as in Figure 3. The main part of the proof is to extract from these sequences of cycles a reachable good multicycle thanks to the characterization of Item 4. (Notice that when ϵ converges to 0, the angle $> 180^o$ made by the vectors of Figure 3 converges to an angle $\geq 180^o$.)

4 Two-player setting

In this section we consider two-player energy mean-payoff games. We show that the four variants of the energy mean-payoff decision problem are in co-NP. To establish this, we show that if the answer to this problem is No, then $\mathcal{P}_2$ has a spoiling memoryless strategy σ_2 that he can use for all initial credits $c_0 \in \mathbb{N}$. In the game structure $G(\sigma_2)$, $\mathcal{P}_1$ is then the only player and we can apply the results of the previous section, in particular Theorem 3. We also show that in case of mean-payoff objectives with strict inequality, the energy mean-payoff decision problem can be reduced to the unknown initial credit problem for 4-dimensional energy games. If follows by [29] that our decision problem can be solved in pseudo-polynomial time and that finite-memory winning strategies with pseudo-polynomial size for $\mathcal{P}_1$ exist and can effectively be constructed. In case of mean-payoff objectives with non-strict inequality, we already know that infinite memory is necessary for $\mathcal{P}_1$ in player-1 energy mean-payoff games by Theorem 3. We show how to construct such strategies. The results that we establish in this section are summarized in the following theorem.

▶ **Theorem 7.** *The energy mean-payoff decision problem for two-player game structures is in co-NP. Moreover,*

- *both problems* $\mathsf{E} \cap \underline{\mathsf{MP}}^{>0}$ *and* $\mathsf{E} \cap \overline{\mathsf{MP}}^{>0}$ *can be solved in pseudo-polynomial time and exponential-memory strategies are sufficient for* $\mathcal{P}_1$ *to win;*
- *for both problems* $\mathsf{E} \cap \underline{\mathsf{MP}}^{\geq 0}$ *and* $\mathsf{E} \cap \overline{\mathsf{MP}}^{\geq 0}$*, in general,* $\mathcal{P}_1$ *needs infinite memory to win.*

In all cases, winning strategies can be effectively constructed for both players.

The proof of this result is detailed in the following sections.

4.1 Memoryless winning strategies for $\mathcal{P}_2$

For all four variants of mean-payoff energy objective, we here establish that $\mathcal{P}_2$ does not need any memory for his winning strategies. Therefore, thanks to Theorem 3, the energy mean-payoff decision problem can be solved in co-NP.

▶ **Proposition 8.** *Let* $\sim \in \{>, \geq\}$*. For all energy mean-payoff games* G *and all initial vertices* v_0*, if the answer to the energy mean-payoff problem* $\mathsf{E} \cap \underline{\mathsf{MP}}^{\sim 0}$ *(resp.* $\mathsf{E} \cap \overline{\mathsf{MP}}^{\sim 0}$*) is* No*, then there exists a memoryless strategy* σ_2 *for* $\mathcal{P}_2$ *such that for all initial credits* $c_0 \in \mathbb{N}$*, no play* ρ *consistent with* σ_2 *from* v_0 *belongs to* $\Omega = \mathsf{Energy}_1(c_0) \cap \underline{\mathsf{MP}}_2(\sim 0)$ *(resp. to* $\Omega = \mathsf{Energy}_1(c_0) \cap \overline{\mathsf{MP}}_2(\sim 0)$*).*

The proof of this proposition is given in [14]. Note that energy objectives are not prefix-independent objectives and as a consequence this proposition does not directly follow from the results of [30] where are given general conditions that guarantee the existence of a memoryless winning strategy for one of the players. However our proof is an adaptation of the proof technique of [9, 21, 27, 30].

Notice that from Theorems 5-6 and Proposition 8, we directly get the following corollary.

▶ **Corollary 9.** *For all energy mean-payoff games* G *and initial vertices* v_0*, let* $\sim \in \{>, \geq\}$*. Then* $\mathcal{P}_1$ *is winning from* v_0 *for* $\mathsf{Energy}_1(c_0) \cap \underline{\mathsf{MP}}_2(\sim 0)$ *for some initial credit* c_0 *if and only if he is winning from* v_0 *for* $\mathsf{Energy}_1(c_0) \cap \overline{\mathsf{MP}}_2(\sim 0)$ *for some initial credit* c_0*.*

While Proposition 8 allows us to obtain the membership in co-NP of the decision problems and to effectively construct winning memoryless strategies for $\mathcal{P}_2$, unfortunately it does not tell us how $\mathcal{P}_1$ must play from a winning vertex (when spoiling strategies do not exist for $\mathcal{P}_2$). In the following two sections we provide results that show how $\mathcal{P}_1$ needs to play in order to win energy mean-payoff games. We first show that $\mathcal{P}_1$ can win with finite memory for the case of strict inequalities, and then we provide infinite-memory winning strategies for the case of non-strict inequalities. For the later case, we already know that infinite memory is necessary even player-1 game structures (see Theorem 3).

4.2 Strategies for $\mathcal{P}_1$: case of strict inequalities

In case of strict inequalities, our solution is based on a reduction to multi-dimensional energy games [18] for which we know how to construct strategies for $\mathcal{P}_1$.

4.2.1 Multi-dimensional energy games

We need to recall the concept of *d-dimensional energy games*, with $d \in \mathbb{N}_0$. Those games are played on d-dimensional game structure $G = (V, V_1, V_2, E, w)$ where the weight function $w : E \to \mathbb{Z}^d$ assigns a d-tuple (instead of a pair) of weights $w(e)$ to each edge $e \in E$. The *unknown initial credit problem* asks, given a d-dimensional game structure and an initial vertex v_0, to decide whether there exists an initial credit $c_0 = (c_{0,1}, \dots, c_{0,d}) \in \mathbb{N}^d$ and a winning strategy for $\mathcal{P}_1$ for the objective $\Omega = \cap_{j=1}^{d} \mathsf{Energy}_j(c_{0,j})$. When $d = 1$ and the answer to this problem is Yes, we denote by $c(v_0) \in \mathbb{N}$ the minimum initial credit for which $\mathcal{P}_1$ has a winning strategy from v_0. The complexity of this problem has been first studied in [18, 21, 35] and then in [29] for a fixed number of dimensions.

▶ **Theorem 10** ([18, 21, 29, 35]). *The unknown initial credit problem for d-dimensional energy games can be solved in pseudo-polynomial time, that is in time $(|V| \cdot ||E||)^{\mathcal{O}(d^4)}$. If the answer to this problem is*

- *Yes, then exponential-memory strategies are sufficient and necessary for player $\mathcal{P}_1$ to win,*
- *No, then $\mathcal{P}_2$ has a spoiling memoryless strategy σ_2 that he can use for all initial credits $c_0 \in \mathbb{N}^d$.*

We recall the next useful lemma.

▶ **Lemma 11** ([17]). *Let G be a 1-dimensional energy game and v_0 be an initial vertex. For all plays ρ consistent with a winning strategy σ_1 for $\mathcal{P}_1$, if the initial credit is $c(v_0) + \Delta$ for $\Delta \geq 0$, then the energy level at all positions of ρ where a state v occurs is at least $c(v) + \Delta$.*

The next proposition shows that we can reduce energy mean-payoff games with strict inequality constraints to energy games with 4 dimensions.

▶ **Proposition 12.** *The problems $\mathsf{E} \cap \underline{\mathsf{MP}}^{>0}$ and $\mathsf{E} \cap \overline{\mathsf{MP}}^{>0}$ for energy mean-payoff games are both polynomially reducible to the unknown initial credit problem for 4-dimensional energy games. Moreover, for the energy game G' constructed from the given G, we have $||E'|| = ||E||$ and $|V'|, |E'|$ are linear in $|V|, |E|$, and from a finite-memory winning strategy σ_1' of $\mathcal{P}_1$ in G', we can derive a finite-memory winning strategy σ_1 of $\mathcal{P}_1$ in G such that the memory size of σ_1 is upper bounded by the memory size of σ_1'.*

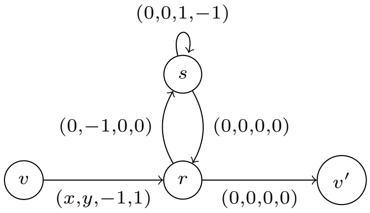


Figure 4 Construction of a 4-dimensional energy game.

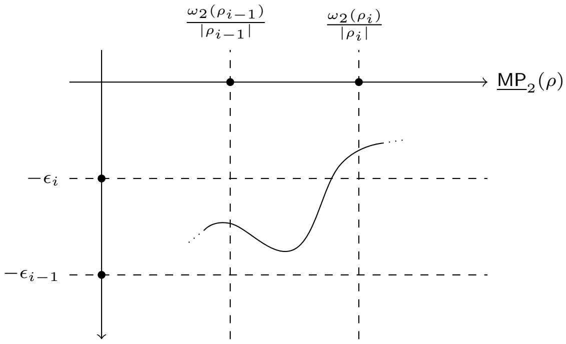


Figure 5 ρ satisfies $\underline{\mathsf{MP}}_2(\rho) \geq 0$.

Proof. We first explain the reduction and then we give the main intuitions that justify the correctness (see [14] for the formal detailed proof). Given an energy mean-payoff game structure $G = (V, V_1, V_2, E, w)$ with $w : E \to \mathbb{Z}^2$, we construct a 4-dimensional energy game $G' = (V', V'_1, V'_2, E', w')$ with $w' : E' \to \mathbb{Z}^4$ as follows. Each edge $e = (v, v') \in E$ labeled by $w(e) = (x, y)$ is replaced by the following gadget composed of:

- five edges $(v, r), (r, s), (s, s), (s, r)$, and (r, v') where r, s are two new vertices,
- such that $w'(v, r) = (x, y, -1, 1)$, $w'(r, s) = (0, -1, 0, 0)$, $w'(s, s) = (0, 0, 1, -1)$, $w'(s, r) = (0, 0, 0, 0)$, and $w'(r, v') = (0, 0, 0, 0)$.

This is illustrated in Figure 4. The set V'_2 is equal to V_2, and V'_1 is composed of all vertices of V_1 and the $2 \cdot |E|$ new vertices (two for each edge of G). By construction, we have $||E'|| = ||E||$ and $|V'|, |E'|$ are linear in $|V|, |E|$. Now, we show that if $\mathcal{P}_i$ wins in G' then $\mathcal{P}_i$ wins in G, for $i \in \{1, 2\}$.

First, assume that $\mathcal{P}_1$ wins in G' with a strategy σ'_1. By Theorem 10, we can assume that σ'_1 is a finite-memory strategy. Let us construct a corresponding strategy σ_1 in G. The graph G' is a structural copy of G where each edge is replaced by the gadget of Figure 4. So to each path π' of G' that ends in a vertex of G (so not in a s or r vertex), there is a corresponding path π in G obtained by removing all the new vertices introduced by the gadget. We construct the strategy σ_1 as follows: the edge taken by σ_i after history π is simply the edge that σ'_i enters in G' after history π'. Let us show that σ_1 is winning in G. Remember that all dimensions in G' are interpreted as energy dimensions. The first dimension which models energy in G is unchanged by the gadget as all weights are 0 for the first dimension in the newly introduced edges. The second dimension, which corresponds to the mean-payoff dimension in G, is transformed into an energy dimension. We make a few remarks. If $\mathcal{P}_1$ decides to go from v to r and then directly to v', the effect on the energy accumulated on the second dimension is the same as in G. Nevertheless because the third dimension is affected negatively by all edges with the exception of the self loops on vertices of type s, it is clear that $\mathcal{P}_1$ needs to take periodically the edges from r-vertices to s-vertices and loop in s in order to recharge the energy on the third dimension. So, the intuition behind our construction is simple: in G', $\mathcal{P}_1$ can play as in G but he needs to recharge periodically dimension three by looping on s. Also, let us note that $\mathcal{P}_1$ always needs to leave the gadget composed of the s and r vertices as otherwise the fourth dimension would go arbitrary low and so this would violate the corresponding energy objectives. Finally, we note that second dimension is decreased when $\mathcal{P}_1$ takes the edge from r to s. So, in order to satisfy the energy objective in G' for dimension two, $\mathcal{P}_1$ needs to accumulate unbounded reward on that dimension in the other edges (and so in the corresponding edges in G). As by hypothesis the strategy σ'_1 is finite-memory, this implies that mean-payoff accumulated on dimension two will be strictly positive when playing the corresponding strategy σ_1 in G. This in turn implies that σ_1 is winning in G.

Assume now that σ'_2 is a winning strategy for $\mathcal{P}_2$ in G'. It can be supposed to be memoryless by Theorem 10. As $V_2 = V'_2$, we can interprete σ'_2 in G thus leading to a player-1 game structure $G(\sigma'_2)$. Similar arguments as done before together with Theorem 5 show that σ'_2 is winning for $\mathcal{P}_2$ in G. ◀

4.3 Strategies for $\mathcal{P}_1$: case of non-strict inequalities

By Theorem 6, we know that infinite memory may be necessary for $\mathcal{P}_1$ to win in case of non-strict inequalities. The reduction to multi-dimensional energy games of previous section is thus not applicable for this case. Instead, we show how we can effectively construct a winning strategy for $\mathcal{P}_1$ by combining an infinite number of finite-memory strategies.

▶ **Proposition 13.** *For both problems $\mathsf{E} \cap \underline{\mathsf{MP}}^{\geq 0}$ and $\mathsf{E} \cap \overline{\mathsf{MP}}^{\geq 0}$, if $\mathcal{P}_1$ is winning from an initial vertex v_0, then one can effectively construct a strategy for him to win from v_0. This strategy requires infinite memory.*

Proof. Remember by Corollary 9 that $\mathcal{P}_1$ is winning from v_0 for the objective $\mathsf{Energy}(c_0) \cap \underline{\mathsf{MP}}(\geq 0)$ for some c_0 if and only if he is winning from v_0 for the objective $\mathsf{Energy}(c_0) \cap \overline{\mathsf{MP}}(\geq 0)$ for some c_0. Here, we show how to construct a winning strategy for $\mathcal{P}_1$ for the mean-payoff-inf case only. Indeed such a winning strategy is also winning for the mean-payoff-sup case.

We first note that if $\mathcal{P}_1$ is winning from a vertex v for the objective $\Omega(c_0) = \mathsf{Energy}_1(c_0) \cap \underline{\mathsf{MP}}_2(\geq 0)$, then he is also winning from v for the objective $\Omega_i(c_0) = \mathsf{Energy}_1(c_0) \cap \underline{\mathsf{MP}}_2(> -\epsilon_i)$ for all $\epsilon_i = \frac{1}{2^i}$, $i \in \mathbb{N}_0$. Let Win be the set of vertices v from which $\mathcal{P}_1$ is winning for $\Omega(c_0)$ for some c_0. In particular $v_0 \in \mathsf{Win}$ by hypothesis. From now on, we assume that the vertices not in Win are removed from V leading to a game structure that we still denote by G. This can be done as a winning strategy for $\mathcal{P}_1$ will never enter those vertices.

For all vertices $v \in \mathsf{Win}$, we denote by $c(v) \in \mathbb{N}$ the minimum initial credit from which $\mathcal{P}_1$ is winning for $\Omega(c(v))$ from v. Similarly for all $i \in \mathbb{N}_0$, we denote by $c_i(v) \in \mathbb{N}$ the minimum initial credit from which he is winning for $\Omega_i(c_i(v))$ from v and by σ_i^v such a winning strategy for $\mathcal{P}_1$. Recall by Proposition 12 that all strategies σ_i^v can be supposed to be finite-memory and to have memory size bounded by M_i^v. The game structure $G(\sigma_i^v)$ induced by σ_i^v has a number of vertices equal to

$$N_i^v = |\mathsf{Win}| \cdot M_i^v \tag{1}$$

Also, we have that $c_1(v) \leq c_2(v) \leq c_3(v) \leq \ldots \leq c(v)$. Moreover as these initial credits are integers,

$$\exists k_v, \forall i \geq k_v : \quad c_i(v) = c_{k_v}(v). \tag{2}$$

Let us define

$$\kappa = \max_{v \in \mathsf{Win}} k_v \text{ and } \gamma = \max\{c_{i+1}(v) - c_i(v) \mid v \in \mathsf{Win}, i \in \mathbb{N}_0\}. \tag{3}$$

These constants will be useful later for the energy objective.

An effective winning strategy for $\mathcal{P}_1$. Let us define a strategy τ_1 for $\mathcal{P}_1$ from v_0 that will be proved to be winning for $\mathcal{P}_1$. A play ρ consistent with τ_i is the limit of a sequence of prefixes ρ_i of increasing length constructed in the following way:

1. Initialize $i = 1$ and $\rho_0 = v_0$;
2. Assume that a prefix ρ_{i-1} has been constructed so far and that its last vertex is v_{i-1}. Apply, starting from v_{i-1}, the strategy $\sigma_i^{v_{i-1}}$ (against $\mathcal{P}_2$) until the produced path π_i consistent with $\sigma_i^{v_{i-1}}$ and the path ρ_i equal to the concatenation ρ_{i-1} with π_i satisfy
$$w_2(\rho_i) > N_{i+1}^{v_i} \cdot ||E|| - |\rho_i| \cdot \epsilon_i. \tag{4}$$
3. Increment i by 1 and goto 2.

Notice that in (4), we require for $w_2(\rho_i)$ more than $w_2(\rho_i) > -|\rho_i| \cdot \epsilon_i$. Indeed the latter inequality would be enough to guarantee that the mean-payoff-sup value of ρ satisfies $\overline{\mathsf{MP}}(\rho) \geq 0$ but we will explain later that we need (4) to guarantee $\underline{\mathsf{MP}}(\rho) \geq 0$.

For the correctness of the given construction, we need to prove that for each $i \in \mathbb{N}_0$, there exists a path ρ_i satisfying (4). This is a consequence of point (ii) of the next lemma.

▶ **Lemma 14.** *As each σ_i^v is a finite-memory strategy from v winning for $\mathsf{Energy}_1(c_0) \cap \underline{\mathsf{MP}}_2(> -\epsilon_i)$,*

(i) *for all plays π consistent with σ_i^v from v, for all $k \in \mathbb{N}$, we have $w_2(\pi[0,k]) > -N_i^v \cdot ||E|| - k \cdot \epsilon_i$, and*

(ii) *for all $K \in \mathbb{N}$, there exists $k \in \mathbb{N}$ such that for all plays π consistent with σ_i^v from v, we have $w_2(\pi[0,k]) > K - k \cdot \epsilon_i$.*

Proof. Let us come back to the game structure $G(\sigma_i^v)$ with N_i^v vertices (by (1)). As σ_i^v is winning for the objective $\underline{\mathsf{MP}}_2(> -\epsilon_i)$, all reachable cycles C in $G(\sigma_i^v)$ have a average weight

$$\frac{w_2(C)}{|C|} > -\epsilon_i. \tag{5}$$

Moreover as the weight $w_2(C)$ is an integer, $w_2(C) \geq -|C| \cdot \epsilon_i + t_C$, for some $t_C > 0$. Let $t = \min\{t_C \mid C \text{ reachable cycle in } G(\sigma_i^v)\}$. This tells us that one unit $t > 0$ of weight is accumulated each time a cycle is closed in $G(\sigma_i^v)$:

$$w_2(C) \geq -|C| \cdot \epsilon_i + t. \tag{6}$$

Let us prove (i). Consider a play π consistent with σ_i^v from v, i.e., an infinite path in $G(\sigma_i^v)$. Let $k \in \mathbb{N}$ and let us reason on the cycle decomposition of $\pi[0,k]$. First, as the acyclic part of this decomposition has a length bounded by N_i^v, its weight is bounded below by $-N_i^v \cdot ||E||$. Second, let ℓ be the total length of the cycles C of the cyclic decomposition of $\pi[0,k]$. As all cycles C in $G(\sigma_i^v)$ satisfy (5), we conclude that the total weight of this cyclic part of $\pi[0,k]$ is bounded below by $-\ell \cdot \epsilon_i$. Finally, as $\ell \leq k$, we obtain the claimed lower bound of (i), that is, $w_2(\pi[0,k]) > -N_i^v \cdot ||E|| - k \cdot \epsilon_i$.

Let us now prove (ii). We simply repeat the arguments given for (i) by using (6) instead of (5). If α cycles are closed during the cycle decomposition of $\pi[0,k]$, we then get $w_2(\pi[0,k]) \geq \alpha \cdot t - N_i^v \cdot ||E|| - k \cdot \epsilon_i$ instead of the inequality of (i). So, given $K \in \mathbb{N}$, take $k \in \mathbb{N}$ such that α is large enough to get an accumulated positive weight $\alpha \cdot t$ such that $\alpha \cdot t - N_i^v \cdot ||E|| > K$. This establishes (ii). ◀

Let us prove that τ_1 is a winning strategy (with infinite memory) from v_0 for the objective $\Omega(d_0)$ with the initial credit

$$d_0 = \kappa \cdot \gamma + c_1(v_0) \tag{7}$$

with the constants of (3). Let ρ be a play consistent with τ_1 from v_0, i.e., ρ is the limit of a sequence of prefixes ρ_i as described previously in the definition of τ_i. Remember that each ρ_i, $i \in \mathbb{N}_0$, is the concatenation of ρ_{i-1} and π_i such that π_i is consistent with $\sigma_i^{v_{i-1}}$ from v_{i-1}.

Mean-payoff-inf objective. We begin by showing that ρ satisfies $\underline{\mathsf{MP}}_2(\rho) \geq 0$. To achieve this goal, it is enough to show that for all $i \in \mathbb{N}_0$, the average weight never falls below $-\epsilon_{i-1}$ during the construction of ρ_i (i.e. the construction of π_i), and this average weight is above $-\epsilon_i$ at the end of the construction of ρ_i (see Figure 5).

Let us show that such a property is a consequence of Lemma 14 and inequality (4) satisfied by ρ_i. First by (4), the average weight of ρ_i satisfies $\frac{w_2(\rho_i)}{|\rho_i|} > -\epsilon_i$. Second, consider any prefix $\pi[0,k]$ of π_i and the corresponding prefix $\rho[0,k']$ of ρ_i such that $k' = k + |\rho_{i-1}|$. Then by point (i) of Lemma 14, we have $w_2(\pi[0,k]) > -N_i^v \cdot ||E|| - k \cdot \epsilon_i$, and by (4) applied to ρ_{i-1}, we have $w_2(\rho_{i-1}) > N_i^{v_i} \cdot ||E|| - |\rho_{i-1}| \cdot \epsilon_{i-1}$. Therefore we get

$$\begin{aligned} w_2(\rho[0,k']) &= w_2(\rho_{i-1}) + w_2(\pi[0,k]) \\ &> (N_i^{v_i} \cdot ||E|| - |\rho_{i-1}| \cdot \epsilon_{i-1}) + (-N_i^v \cdot ||E|| - k \cdot \epsilon_i) \\ &> -|\rho[0,k']| \cdot \epsilon_{i-1} \end{aligned}$$

Hence, as announced, the average weight of the prefix $\rho[0,k']$ of ρ_i is above $-\epsilon_{i-1}$.

Energy objective. It remains to explain why the energy objective is also satisfied by ρ with the initial credit d_0 defined in (7). Recall from the definition of τ_1 that ρ is the limit of a sequence of prefixes ρ_i such that each ρ_i is the concatenation of ρ_{i-1} and π_i. Recall also that $c_i(v) \in \mathbb{N}$ is the minimum initial credit for which σ_i^v is winning from v.

By construction, π_1 is consistent with $\sigma_1^{v_0}$ with the initial credit $d_0 = c_1(v_0) + \Delta_1$, where $\Delta_1 = \kappa \cdot \gamma$. Hence the energy level of $\rho_1 = \pi_1$ never drops below zero and it is at least equal to $c_1(v_1) + \Delta_1$ in the last vertex v_1 of ρ_1 by Lemma 11. Similarly π_2 is consistent with $\sigma_2^{v_1}$ with the initial credit $c_1(v_1) + \Delta_1 = c_2(v_1) + \Delta_2$, where $\Delta_2 = \kappa \cdot \gamma - (c_2(v_1) - c_1(v_1))$. Hence the energy level of ρ_2 never drops below zero and it is at least equal to $c_2(v_2) + \Delta_2$ in the last vertex v_2 of ρ_2 by Lemma 11. This argument can be repeated for all $i \in \mathbb{N}_0$: the energy level of ρ_i never drops below zero and it is at least equal to $c_i(v_i) + \Delta_i$, with $\Delta_i = \kappa \cdot \gamma - \sum_{j=1}^{i-1}(c_{j+1}(v_j) - c_j(v_j))$. Notice that we always have $\Delta_i \geq 0$ by (2) and by definition of κ and γ (see (3)). Therefore the energy level of ρ never drops belows zero.

This proves that τ_1 is a winning strategy for the objective $\mathsf{Energy}_1(d_0) \cap \underline{\mathsf{MP}}_2(\geq 0)$ and thus conclude the proof. ◀

4.4 Proof of Theorem 7

We conclude this section with the proof of Theorem 7.

Proof of Theorem 7. We establish the three assertions of the theorem as follows.

We first prove that the energy mean-payoff decision problems for two-player games G are in co-NP for the four variants. This result is obtained as follows. By Proposition 8, memoryless strategies are sufficient for $\mathcal{P}_2$ to win, for all four variants. Hence, the following is an algorithm in co-NP: guess a memoryless strategy σ_2 for $\mathcal{P}_2$, and in the resulting one-player game $G(\sigma_2)$, verify in polynomial time whether $\mathcal{P}_1$ is winning thanks to Theorem 3.

Second, we consider the two variants with strict inequalities. By Proposition 12, there exists a polynomial reduction of the energy mean-payoff decision problem to the unknown initial credit problem for 4-dimensional energy games. By Theorem 10, it follows that the energy mean-payoff decision problem can be solved in pseudo-polynomial time and that exponential-memory strategies are sufficient for $\mathcal{P}_1$ to win.

Finally, we consider the last two variants with non-strict inequalities. In Proposition 13, we have shown how we can effectively construct a winning strategy for $\mathcal{P}_1$ in this case. ◀

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Piotr Hofman, Richard Mayr, K. Narayan Kumar, and Patrick Totzke. Infinite-state energy games. 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 7:1–7:10. ACM, 2014. doi:10.1145/2603088.2603100.

2 Parosh Aziz Abdulla, Richard Mayr, Arnaud Sangnier, and Jeremy Sproston. Solving Parity Games on Integer Vectors. In Pedro R. D'Argenio and Hernán C. Melgratti, editors, *CONCUR 2013 - Concurrency Theory - 24th International Conference, CONCUR 2013, Buenos Aires, Argentina, August 27-30, 2013, Proceedings*, volume 8052 of *Lecture Notes in Computer Science*, pages 106–120. Springer, 2013. doi:10.1007/978-3-642-40184-8_9.

3 Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. Graph Games and Reactive Synthesis. In Edmund M. Clarke, Thomas A. Henzinger, Helmut Veith, and Roderick Bloem, editors, *Handbook of Model Checking*, pages 921–962. Springer, 2018.

4 Udi Boker, Thomas A. Henzinger, and Arjun Radhakrishna. Battery transition systems. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 595–606. ACM, 2014. doi:10.1145/2535838.2535875.

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, Nicolas Markey, Mickael Randour, Kim G. Larsen, and Simon Laursen. Average-energy games. *Acta Inf.*, 55(2):91–127, 2018. doi:10.1007/s00236-016-0274-1.

8 Tomás Brázdil, Krishnendu Chatterjee, Antonín Kucera, and Petr Novotný. Efficient Controller Synthesis for Consumption Games with Multiple Resource Types. In P. Madhusudan and Sanjit A. Seshia, editors, *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 23–38. Springer, 2012. doi:10.1007/978-3-642-31424-7_8.

9 Tomás Brázdil, Petr Jancar, and Antonín Kucera. Reachability Games on Extended Vector Addition Systems with States. *CoRR*, abs/1002.2557, 2010. arXiv:1002.2557.

10 Tomás Brázdil, David Klaska, Antonín Kucera, and Petr Novotný. Minimizing Running Costs in Consumption Systems. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014, Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 457–472. Springer, 2014. doi:10.1007/978-3-319-08867-9_30.

11 Tomás Brázdil, Antonín Kucera, and Petr Novotný. Optimizing the Expected Mean Payoff in Energy Markov Decision Processes. In Cyrille Artho, Axel Legay, and Doron Peled, editors, *Automated Technology for Verification and Analysis - 14th International Symposium, ATVA 2016, Chiba, Japan, October 17-20, 2016, Proceedings*, volume 9938 of *Lecture Notes in Computer Science*, pages 32–49, 2016. doi:10.1007/978-3-319-46520-3_3.

12 Lubos Brim, Jakub Chaloupka, Laurent Doyen, Raffaella Gentilini, and Jean-François Raskin. Faster algorithms for mean-payoff games. *Formal Methods in System Design*, 38(2):97–118, 2011. doi:10.1007/s10703-010-0105-x.

13 Véronique Bruyère. Computer Aided Synthesis: A Game-Theoretic Approach. In Émilie Charlier, Julien Leroy, and Michel Rigo, editors, *Developments in Language Theory - 21st International Conference, DLT 2017, Liège, Belgium, August 7-11, 2017, Proceedings*, volume 10396 of *Lecture Notes in Computer Science*, pages 3–35. Springer, 2017. doi:10.1007/978-3-319-62809-7_1.

14 Véronique Bruyère, Quentin Hautem, Mickael Randour, and Jean-François Raskin. Energy mean-payoff games. *CoRR*, abs/1907.01359, 2019. arXiv:1907.01359.

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 Arindam Chakrabarti, Luca de Alfaro, Thomas A. Henzinger, and Mariëlle Stoelinga. Resource Interfaces. In Rajeev Alur and Insup Lee, editors, *Embedded Software, Third International Conference, EMSOFT 2003, Philadelphia, PA, USA, October 13-15, 2003, Proceedings*, volume 2855 of *Lecture Notes in Computer Science*, pages 117–133. Springer, 2003. doi:10.1007/978-3-540-45212-6_9.

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, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. Generalized Mean-payoff and Energy Games. In Kamal Lodaya and Meena Mahajan, editors, *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2010, December 15-18, 2010, Chennai, India*, volume 8 of *LIPIcs*, pages 505–516. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010. doi:10.4230/LIPIcs.FSTTCS.2010.505.

19 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.

20 Krishnendu Chatterjee, Thomas A. Henzinger, and Marcin Jurdzinski. Mean-Payoff Parity Games. In *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pages 178–187. IEEE Computer Society, 2005. doi:10.1109/LICS.2005.26.

21 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.

22 Krishnendu Chatterjee and Yaron Velner. The Complexity of Mean-Payoff Pushdown Games. *J. ACM*, 64(5):34:1–34:49, 2017. doi:10.1145/3121408.

23 Thomas Colcombet, Marcin Jurdzinski, Ranko Lazic, and Sylvain Schmitz. Perfect half space games. In *LICS Proceedings*, pages 1–11. IEEE Computer Society, 2017. doi:10.1109/LICS.2017.8005105.

24 Laure Daviaud, Marcin Jurdzinski, and Ranko Lazic. A pseudo-quasi-polynomial algorithm for mean-payoff parity games. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 325–334. ACM, 2018. doi:10.1145/3209108.3209162.

25 A. Ehrenfeucht and J. Mycielski. Positional strategies for mean payoff games. *International Journal of Game Theory*, 8:109–113, 1979. doi:10.1007/BF01768705.

26 Uli Fahrenberg, Line Juhl, Kim G. Larsen, and Jirí Srba. Energy Games in Multiweighted Automata. In Antonio Cerone and Pekka Pihlajasaari, editors, *Theoretical Aspects of Computing - ICTAC 2011 - 8th International Colloquium, Johannesburg, South Africa, August 31 - September 2, 2011, Proceedings*, volume 6916 of *Lecture Notes in Computer Science*, pages 95–115. Springer, 2011. doi:10.1007/978-3-642-23283-1_9.

27 Hugo Gimbert and Wieslaw Zielonka. Games Where You Can Play Optimally Without Any Memory. In Martín Abadi and Luca de Alfaro, editors, *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, volume 3653 of *Lecture Notes in Computer Science*, pages 428–442. Springer, 2005. doi:10.1007/11539452_33.

28 Line Juhl, Kim Guldstrand Larsen, and Jean-François Raskin. Optimal Bounds for Multiweighted and Parametrised Energy Games. In Zhiming Liu, Jim Woodcock, and Huibiao Zhu, editors, *Theories of Programming and Formal Methods - Essays Dedicated to Jifeng He on the Occasion of His 70th Birthday*, volume 8051 of *Lecture Notes in Computer Science*, pages 244–255. Springer, 2013. doi:10.1007/978-3-642-39698-4_15.

29 Marcin Jurdzinski, Ranko Lazic, and Sylvain Schmitz. Fixed-Dimensional Energy Games are in Pseudo-Polynomial Time. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 2015. doi:10.1007/978-3-662-47666-6_21.

30 Eryk Kopczynski and Damian Niwinski. A simple indeterminate infinite game. In Vasco Brattka, Hannes Diener, and Dieter Spreen, editors, *Logic, Computation, Hierarchies*, volume 4 of *Ontos Mathematical Logic*, pages 205–212. De Gruyter, 2014. doi:10.1515/9781614518044.205.

31 S. Rao Kosaraju and Gregory F. Sullivan. Detecting Cycles in Dynamic Graphs in Polynomial Time (Preliminary Version). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 398–406. ACM, 1988. doi:10.1145/62212.62251.

32 Antonín Kucera. Playing Games with Counter Automata. In Alain Finkel, Jérôme Leroux, and Igor Potapov, editors, *Reachability Problems - 6th International Workshop, RP 2012, Bordeaux, France, September 17-19, 2012, Proceedings*, volume 7550 of *Lecture Notes in Computer Science*, pages 29–41. Springer, 2012. doi:10.1007/978-3-642-33512-9_4.

33 Amir Pnueli and Roni Rosner. On the Synthesis of an Asynchronous Reactive Module. In Giorgio Ausiello, Mariangiola Dezani-Ciancaglini, and Simona Ronchi Della Rocca, editors, *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.

34 Yaron Velner. Robust Multidimensional Mean-Payoff Games are Undecidable. In Andrew M. Pitts, editor, *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 312–327. Springer, 2015. doi:10.1007/978-3-662-46678-0_20.

35 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.

36 U. Zwick and M. Paterson. The Complexity of Mean Payoff Games on Graphs. *Theoretical Computer Science*, 158(1-2):343–359, 1996.

Equilibrium Design for Concurrent Games

Julian Gutierrez
Department of Computer Science, University of Oxford, UK
julian.gutierrez@cs.ox.ac.uk

Muhammad Najib
Department of Computer Science, University of Oxford, UK
mnajib@cs.ox.ac.uk

Giuseppe Perelli
Department of Computer Science, University of Göteborg, Sweden
giuseppe.perelli@gu.se

Michael Wooldridge
Department of Computer Science, University of Oxford, UK
michael.wooldridge@cs.ox.ac.uk

Abstract

In game theory, *mechanism design* is concerned with the design of incentives so that a desired outcome of the game can be achieved. In this paper, we study the design of incentives so that a desirable equilibrium is obtained, for instance, an equilibrium satisfying a given temporal logic property – a problem that we call *equilibrium design*. We base our study on a framework where system specifications are represented as temporal logic formulae, games as quantitative concurrent game structures, and players' goals as mean-payoff objectives. In particular, we consider system specifications given by LTL and GR(1) formulae, and show that implementing a mechanism to ensure that a given temporal logic property is satisfied on some/every Nash equilibrium of the game, whenever such a mechanism exists, can be done in PSPACE for LTL properties and in NP/Σ_2^{P} for GR(1) specifications. We also study the complexity of various related decision and optimisation problems, such as optimality and uniqueness of solutions, and show that the complexities of all such problems lie within the polynomial hierarchy. As an application, equilibrium design can be used as an alternative solution to the rational synthesis and verification problems for concurrent games with mean-payoff objectives whenever no solution exists, or as a technique to repair, whenever possible, concurrent games with undesirable rational outcomes (Nash equilibria) in an optimal way.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Computing methodologies → Multi-agent systems; Theory of computation → Algorithmic game theory

Keywords and phrases Games, Temporal logic, Synthesis, Model checking, Nash equilibrium

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.22

Acknowledgements Najib acknowledges the financial support of the Indonesia Endowment Fund for Education (LPDP), and Perelli the support of the project "dSynMA", funded by the ERC under the European Union's Horizon 2020 research and innovation programme (grant agreement No 772459).

1 Introduction

Over the past decade, there has been increasing interest in the use of game-theoretic equilibrium concepts such as Nash equilibrium in the analysis of concurrent and multi-agent systems (see, e.g., [3, 4, 8, 14, 15, 17, 23]). This work views a concurrent system as a game, with system components (agents) corresponding to players in the game, which are assumed to be acting rationally in pursuit of their individual preferences. Preferences may be specified by associating with each player a temporal logic goal formula, which the player desires to see satisfied, or by assuming that players receive rewards in each state the system visits, and seek to maximise the average reward they receive (the *mean payoff*). A further

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 22; pp. 22:1–22:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

possibility is to combine goals and rewards: players primarily seek the satisfaction of their goal, and only secondarily seek to maximise their mean payoff. The key decision problems in such settings relate to what temporal logic properties hold on computations of the system that may be generated by players choosing strategies that form a game-theoretic (Nash) equilibrium. These problems are typically computationally complex, since they subsume temporal logic synthesis [32]. If players have LTL goals, for example, then checking whether an LTL formula holds on some Nash equilibrium path in a concurrent game is 2EXPTIME-complete [14, 16, 17], rather than only PSPACE-complete as it is the case for model checking, certainly a computational barrier for the practical analysis and automated verification of reactive, concurrent, and multi-agent systems modelled as multi-player games.

Within this game-theoretic reasoning framework, a key issue is that individually rational choices can cause outcomes that are highly undesirable, and concurrent games also fall prey to this problem. This has motivated the development of techniques for modifying games, in order to avoid bad equilibria, or to facilitate good equilibria. *Mechanism design* is the problem of designing a game such that, if players behave rationally, then a desired outcome will be obtained [26]. Taxation and subsidy schemes are probably the most important class of techniques used in mechanism design. They work by levying taxes on certain actions (or providing subsidies), thereby incentivising players away from some outcomes towards others. The present paper studies the design of subsidy schemes (incentives) for concurrent games, so that a desired outcome (a Nash equilibrium in the game) can be obtained – a problem that we call *Equilibrium design.* We model agents as synchronously executing concurrent processes, with each agent receiving an integer payoff for every state the overall system visits; the overall payoff an agent receives over an infinite computation path is then defined to be the mean payoff over this path. While agents (naturally) seek to maximise their individual mean payoff, the designer of the subsidy scheme wishes to see some temporal logic formula satisfied, either on some or on every Nash equilibrium of the game.

With this model, we assume that the designer – an external principal – has a finite budget that is available for making subsidies, and this budget can be allocated across agent/state pairs. By allocating this budget appropriately, the principal can incentivise players away from some states and towards others. Since the principal has some temporal logic goal formula, it desires to allocate subsidies so that players are rationally incentivised to choose strategies so that the principal's temporal logic goal formula is satisfied in the path that would result from executing the strategies. For this general problem, following [24], we identify two variants of the principal's mechanism design problem, which we refer to as Weak Implementation and Strong Implementation. In the Weak variant, we ask whether the principal can allocate the budget so that the goal is achieved on *some* computation path that would be generated by Nash equilibrium strategies in the resulting system; in the Strong variation, we ask whether the principal can allocate the budget so that the resulting system has at least one Nash equilibrium, and moreover the temporal logic goal is satisfied on *all* paths that could be generated by Nash equilibrium strategies. For these two problems, we consider goals specified by LTL formulae or GR(1) formulae [5], give algorithms for each case, and classify the complexity of the problem. While LTL is a natural language for the specification of properties of concurrent and multi-agent systems, GR(1) is an LTL fragment that can be used to easily express several prefix-independent properties of computation paths of reactive systems, such as ω-regular properties often used in automated formal verification. We then go on to study variations of these two problems, for example considering *optimality* and *uniqueness* of solutions, and show that the complexities of all such problems lie within the polynomial hierarchy, thus making them potentially amenable to efficient practical implementations. Table 1 summarises the main computational complexity results in the paper.

Table 1 Summary of main complexity results.

	LTL Spec.	GR(1) Spec.
Weak Implementation	PSPACE-complete (Thm. 6)	NP-complete (Thm. 7)
Strong Implementation	PSPACE-complete (Cor. 9)	Σ_2^{P}-complete (Thm. 10)
Opt-WI	FPSPACE-complete (Thm. 14)	$\mathsf{FP}^{\mathsf{NP}}$-complete (Thm. 16)
Opt-SI	FPSPACE-complete (Thm. 22)	$\mathsf{FP}^{\Sigma_2^{\mathsf{P}}}$-complete (Thm. 25)
Exact-WI	PSPACE-complete (Cor. 15)	D^{P}-complete (Cor. 17)
Exact-SI	PSPACE-complete (Cor. 23)	$\mathsf{D}_2^{\mathsf{P}}$-complete (Cor. 26)
UOpt-WI	PSPACE-complete (Cor. 18)	Δ_2^{P}-complete (Cor. 19)
UOpt-SI	PSPACE-complete (Cor. 27)	Δ_3^{P}-complete (Cor. 28)

2 Preliminaries

Linear Temporal Logic. LTL [31] extends classical propositional logic with two operators, $\mathbf{X}$ ("next") and $\mathbf{U}$ ("until"), that can be used to express properties of paths. The syntax of LTL is defined with respect to a set AP of atomic propositions as follows:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathbf{X}\varphi \mid \varphi\, \mathbf{U}\, \varphi$$

where $p \in \mathrm{AP}$. As commonly found in the LTL literature, we use of the following abbreviations: $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$, $\varphi_1 \to \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$, $\mathbf{F}\varphi \equiv \top\, \mathbf{U}\, \varphi$, and $\mathbf{G}\varphi \equiv \neg\mathbf{F}\neg\varphi$.

We interpret formulae of LTL with respect to pairs (α, t), where $\alpha \in (2^{\mathrm{AP}})^{\omega}$ is an infinite sequence of atomic proposition evaluations that indicates which propositional variables are true in every time point and $t \in \mathbb{N}$ is a temporal index into α. Formally, the semantics of LTL formulae is given by the following rules:

$$\begin{array}{lll}
(\alpha,t) \models \top & & \\
(\alpha,t) \models p & \text{iff} & p \in \alpha_t \\
(\alpha,t) \models \neg\varphi & \text{iff} & \text{it is not the case that } (\alpha,t) \models \varphi \\
(\alpha,t) \models \varphi \vee \psi & \text{iff} & (\alpha,t) \models \varphi \text{ or } (\alpha,t) \models \psi \\
(\alpha,t) \models \mathbf{X}\varphi & \text{iff} & (\alpha,t+1) \models \varphi \\
(\alpha,t) \models \varphi\, \mathbf{U}\, \psi & \text{iff} & \text{for some } t' \geq t : \big((\alpha,t') \models \psi \text{ and} \\
 & & \quad \text{for all } t \leq t'' < t' : (\alpha,t'') \models \varphi\big).
\end{array}$$

If $(\alpha, 0) \models \varphi$, we write $\alpha \models \varphi$ and say that α *satisfies* φ.

General Reactivity of rank 1. The language of *General Reactivity of rank 1*, denoted GR(1), is the fragment of LTL given by formulae written in the following form [5]:

$$(\mathbf{GF}\psi_1 \wedge \ldots \wedge \mathbf{GF}\psi_m) \to (\mathbf{GF}\varphi_1 \wedge \ldots \wedge \mathbf{GF}\varphi_n),$$

where each subformula ψ_i and φ_i is a Boolean combination of atomic propositions.

Mean-Payoff. For a sequence $r \in \mathbb{R}^\omega$, let $\mathsf{mp}(r)$ be the *mean-payoff* value of r, that is,

$$\mathsf{mp}(r) = \lim \inf_{n \to \infty} \mathsf{avg}_n(r)$$

where, for $n \in \mathbb{N} \setminus \{0\}$, we define $\mathsf{avg}_n(r) = \frac{1}{n} \sum_{j=0}^{n-1} r_j$, with r_j the $(j+1)$th element of r.

Arenas. An *arena* is a tuple $A = \langle \mathrm{N}, \mathrm{Ac}, \mathrm{St}, s_0, \mathsf{tr}, \lambda \rangle$ where N, Ac, and St are finite non-empty sets of *players* (write $N = |\mathrm{N}|$), *actions*, and *states*, respectively; if needed, we write $\mathrm{Ac}_i(s)$, to denote the set of actions available to player i at s; $s_0 \in \mathrm{St}$ is the *initial state*; $\mathsf{tr} : \mathrm{St} \times \vec{\mathrm{Ac}} \to \mathrm{St}$ is a *transition function* mapping each pair consisting of a state $s \in \mathrm{St}$ and an *action profile* $\vec{\mathsf{a}} \in \vec{\mathrm{Ac}} = \mathrm{Ac}^{\mathrm{N}}$, one for each player, to a successor state; and $\lambda : \mathrm{St} \to 2^{\mathrm{AP}}$ is a labelling function, mapping every state to a subset of *atomic propositions.*

We sometimes call an action profile $\vec{\mathsf{a}} = (\mathsf{a}_1, \ldots, \mathsf{a}_n) \in \vec{\mathrm{Ac}}$ a *decision*, and denote a_i the action taken by player i. We also consider *partial* decisions. For a set of players $C \subseteq \mathrm{N}$ and action profile $\vec{\mathsf{a}}$, we let $\vec{\mathsf{a}}_C$ and $\vec{\mathsf{a}}_{-C}$ be two tuples of actions, respectively, one for all players in C and one for all players in $\mathrm{N} \setminus C$. We also write $\vec{\mathsf{a}}_i$ for $\vec{\mathsf{a}}_{\{i\}}$ and $\vec{\mathsf{a}}_{-i}$ for $\vec{\mathsf{a}}_{\mathrm{N}\setminus\{i\}}$. For two decisions $\vec{\mathsf{a}}$ and $\vec{\mathsf{a}}'$, we write $(\vec{\mathsf{a}}_C, \vec{\mathsf{a}}'_{-C})$ to denote the decision where the actions for players in C are taken from $\vec{\mathsf{a}}$ and the actions for players in $\mathrm{N} \setminus C$ are taken from $\vec{\mathsf{a}}'$.

A *path* $\pi = (s_0, \vec{\mathsf{a}}^0), (s_1, \vec{\mathsf{a}}^1) \cdots$ is an infinite sequence in $(\mathrm{St} \times \vec{\mathrm{Ac}})^\omega$ such that $\mathsf{tr}(s_k, \vec{\mathsf{a}}^k) = s_{k+1}$ for all k. Paths are generated in the arena by each player i selecting a *strategy* σ_i that will define how to make choices over time. We model strategies as finite state machines with output. Formally, for arena A, a strategy $\sigma_i = (Q_i, q_i^0, \delta_i, \tau_i)$ for player i is a finite state machine with output (a transducer), where Q_i is a finite and non-empty set of *internal states*, q_i^0 is the *initial state*, $\delta_i : Q_i \times \vec{\mathrm{Ac}} \to Q_i$ is a deterministic *internal transition function*, and $\tau_i : Q_i \to \mathrm{Ac}_i$ an *action function.* Let Str_i be the set of strategies for player i. Note that this definition implies that strategies have *perfect information*[1] and finite memory (although we impose no bounds on memory size).

A *strategy profile* $\vec{\sigma} = (\sigma_1, \ldots, \sigma_n)$ is a vector of strategies, one for each player. As with actions, $\vec{\sigma}_i$ denotes the strategy assigned to player i in profile $\vec{\sigma}$. Moreover, by $(\vec{\sigma}_B, \vec{\sigma}'_C)$ we denote the combination of profiles where players in disjoint B and C are assigned their corresponding strategies in $\vec{\sigma}$ and $\vec{\sigma}'$, respectively. Once a state s and profile $\vec{\sigma}$ are fixed, the game has an *outcome*, a path in A, denoted by $\pi(\vec{\sigma}, s)$. Because strategies are deterministic, $\pi(\vec{\sigma}, s)$ is the unique path induced by $\vec{\sigma}$, that is, the sequence $s_0, s_1, s_2, \ldots$ such that

- $s_{k+1} = \mathsf{tr}(s_k, (\tau_1(q_1^k), \ldots, \tau_n(q_n^k)))$, and
- $q_i^{k+1} = \delta_i(s_i^k, (\tau_1(q_1^k), \ldots, \tau_n(q_n^k)))$, for all $k \geq 0$.

Furthermore, we simply write $\pi(\vec{\sigma})$ for $\pi(\vec{\sigma}, s_0)$.

Arenas define the dynamic structure of games, but lack a central aspect of a game: preferences, which give games their strategic structure. A *multi-player game* is obtained from an arena A by associating each player with a goal. We consider multi-player games with mp goals. A multi-player mp game is a tuple $\mathcal{G} = \langle A, (\mathsf{w}_i)_{i \in \mathrm{N}} \rangle$, where A is an arena and $\mathsf{w}_i : \mathrm{St} \to \mathbb{Z}$ is a function mapping, for every player i, every state of the arena into an integer number. In any game with arena A, a path π in A induces a sequence $\lambda(\pi) = \lambda(s_0)\lambda(s_1) \cdots$ of sets of atomic propositions; if, in addition, A is the arena of an mp game, then, for each player i, the sequence $\mathsf{w}_i(\pi) = \mathsf{w}_i(s_0)\mathsf{w}_i(s_1) \cdots$ of weights is also induced. Unless stated otherwise, for a game $\mathcal{G}$ and a path π in it, the payoff of player i is $\mathsf{pay}_i(\pi) = \mathsf{mp}(\mathsf{w}_i(\pi))$.

[1] Mean-payoff games with imperfect information are generally undecidable [13].

Nash equilibrium. Using payoff functions, we can define the game-theoretic concept of Nash equilibrium [26]. For a multi-player game $\mathcal{G}$, a strategy profile $\vec{\sigma}$ is a *Nash equilibrium* of $\mathcal{G}$ if, for every player i and strategy σ'_i for player i, we have

$$\mathsf{pay}_i(\pi(\vec{\sigma})) \geq \mathsf{pay}_i(\pi((\vec{\sigma}_{-i}, \sigma'_i))) \ .$$

Let $\mathrm{NE}(\mathcal{G})$ be the set of Nash equilibria of $\mathcal{G}$.

3 From Mechanism Design to Equilibrium Design

We now describe the two main problems that are our focus of study. As discussed in the introduction, such problems are closely related to the well-known problem of *mechanism design* in game theory. Consider a system populated by agents N, where each agent $i \in \mathrm{N}$ wants to maximise its payoff $\mathsf{pay}_i(\cdot)$. As in a mechanism design problem, we assume there is an external *principal* who has a goal φ that it wants the system to satisfy, and to this end, wants to incentivise the agents to act collectively and rationally so as to bring about φ. In our model, incentives are given by *subsidy schemes* and goals by temporal logic formulae.

Subsidy Schemes. A subsidy scheme defines additional imposed rewards over those given by the weight function w. While the weight function w is fixed for any given game, the principal is assumed to be at liberty to define a subsidy scheme as they see fit. Since agents will seek to maximise their overall rewards, the principal can incentivise agents away from performing visiting some states and towards visiting others; if the principal designs the subsidy scheme correctly, the agents are incentivised to choose a strategy profile $\vec{\sigma}$ such that $\pi(\vec{\sigma}) \models \varphi$. Formally, we model a subsidy scheme as a function $\kappa : \mathrm{N} \to \mathrm{St} \to \mathbb{N}$, where the intended interpretation is that $\kappa(i)(s)$ is the subsidy in the form of a natural number $k \in \mathbb{N}$ that would be imposed on player i if such a player visits state $s \in \mathrm{St}$. For instance, if we have $\mathsf{w}_i(s) = 1$ and $\kappa(i)(s) = 2$, then player i gets $1 + 2 = 3$ for visiting such a state. For simplicity, hereafter we write $\kappa_i(s)$ instead of $\kappa(i)(s)$ for the subsidy for player i.

Notice that having an unlimited fund for a subsidy scheme would make some problems trivial, as the principal can always incentivise players to satisfy φ (provided that there is a path in A satisfying φ). A natural and more interesting setting is that the principal is given a constraint in the form of *budget* $\beta \in \mathbb{N}$. The principal then can only spend within the budget limit. To make this clearer, we first define the *cost* of a subsidy scheme κ as follows.

▶ **Definition 1.** *Given a game $\mathcal{G}$ and subsidy scheme κ, we let* $\mathsf{cost}(\kappa) = \sum_{i \in \mathrm{N}} \sum_{s \in \mathrm{St}} \kappa_i(s)$.

We say that a subsidy scheme κ is *admissible* if it does not exceed the budget β, that is, if $\mathsf{cost}(\kappa) \leq \beta$. Let $\mathcal{K}(\mathcal{G}, \beta)$ denote the set of admissible subsidy schemes over $\mathcal{G}$ given budget $\beta \in \mathbb{N}$. Thus we know that for each $\kappa \in \mathcal{K}(\mathcal{G}, \beta)$ we have $\mathsf{cost}(\kappa) \leq \beta$. We write $(\mathcal{G}, \kappa)$ to denote the resulting game after the application of subsidy scheme κ on game $\mathcal{G}$. Formally, we define the application of some subsidy scheme on a game as follows.

▶ **Definition 2.** *Given a game $\mathcal{G} = \langle A, (\mathsf{w}_i)_{i \in \mathrm{N}} \rangle$ and an admissible subsidy scheme κ, we define $(\mathcal{G}, \kappa) = \langle A, (\mathsf{w}'_i)_{i \in \mathrm{N}} \rangle$, where $\mathsf{w}'_i(s) = \mathsf{w}_i(s) + \kappa_i(s)$, for each $i \in \mathrm{N}$ and $s \in \mathrm{St}$.*

We now come to the main question(s) that we consider in the remainder of the paper. We ask whether the principal can find a subsidy scheme that will incentivise players to collectively choose a rational outcome (a Nash equilibrium) that satisfies its temporal logic goal φ. We call this problem *equilibrium design*. Following [24], we define two variants of this problem, a *weak* and a *strong* implementation of the equilibrium design problem. The formal definition of the problems and the analysis of their respective computational complexity are presented in the next sections.

4 Equilibrium Design: Weak Implementation

In this section, we study the weak implementation of the equilibrium design problem, a logic-based computational variant of the principal's mechanism design problem in game theory. We assume that the principal has full knowledge of the game $\mathcal{G}$ under consideration, that is, the principal uses all the information available of $\mathcal{G}$ to find the appropriate subsidy scheme, if such a scheme exists. We now formally define the weak variant of the implementation problem, and study its respective computational complexity, first with respect to goals (specifications) given by LTL formulae and then with respect to $\mathsf{GR}(1)$ formulae.

Let $\mathrm{WI}(\mathcal{G}, \varphi, \beta)$ denote the set of subsidy schemes over $\mathcal{G}$ given budget β that satisfy a formula φ in at least one path π generated by $\vec{\sigma} \in \mathrm{NE}(\mathcal{G})$. Formally

$$\mathrm{WI}(\mathcal{G}, \varphi, \beta) = \{\kappa \in \mathcal{K}(\mathcal{G}, \beta) : \exists \vec{\sigma} \in \mathrm{NE}(\mathcal{G}, \kappa) \text{ s.t. } \pi(\vec{\sigma}) \models \varphi\}.$$

▶ **Definition 3** (Weak Implementation). *Given a game $\mathcal{G}$, formula φ, and budget β:*

Is it the case that $\mathrm{WI}(\mathcal{G}, \varphi, \beta) \neq \varnothing$?

In order to solve Weak Implementation, we first characterise the Nash equilibria of a multi-player concurrent game in terms of punishment strategies. To do this in our setting, we recall the notion of secure values for mean-payoff games [33].

For a player i and a state $s \in \mathrm{St}$, by $\mathtt{pun}_i(s)$ we denote the punishment value of i over s, that is, the maximum payoff that i can achieve from s, when all other players behave adversarially. Such a value can be computed by considering the corresponding two-player zero-sum mean-payoff game [35]. Thus, it is in $\mathsf{NP} \cap \mathsf{coNP}$, and note that both player i and coalition $\mathrm{N} \setminus \{i\}$ can achieve the optimal value of the game using *memoryless* strategies. Then, for a player i and a value $z \in \mathbb{R}$, a pair $(s, \vec{\mathsf{a}})$ is z-secure for player i if $\mathtt{pun}_i(\mathsf{tr}(s, (\vec{\mathsf{a}}_{-i}, \mathsf{a}'_i))) \leq z$ for every $\mathsf{a}'_i \in \mathrm{Ac}$. Write $\mathtt{pun}_i(\mathcal{G})$ for the set of punishment values for player i in $\mathcal{G}$.

▶ **Theorem 4.** *For every mp game $\mathcal{G}$ and ultimately periodic path $\pi = (s_0, \vec{\mathsf{a}}_0), (s_1, \vec{\mathsf{a}}^1), \ldots$, the following are equivalent:*

1. *There is $\vec{\sigma} \in \mathrm{NE}(\mathcal{G})$ such that $\pi = \pi(\vec{\sigma}, s_0)$;*
2. *There exists $z \in \mathbb{R}^{\mathrm{N}}$, where $z_i \in \mathtt{pun}_i(\mathcal{G})$ such that, for every $i \in \mathrm{N}$*
   - **a.** *for all $k \in \mathbb{N}$, the pair $(s_k, \vec{\mathsf{a}}^k)$ is z_i-secure for i, and*
   - **b.** *$z_i \leq \mathsf{pay}_i(\pi)$.*

The characterisation of Nash Equilibria provided in Theorem 4 will allow us to turn the Weak Implementation problem into a *path finding* problem over $(\mathcal{G}, \kappa)$. On the other hand, with respect to the budget β that the principal has at its disposal, the definition of subsidy scheme function κ implies that the size of $\mathcal{K}(\mathcal{G}, \beta)$ is bounded, and particularly, it is bounded by β and the number of agents and states in the game $\mathcal{G}$, in the following way.

▶ **Proposition 5.** *Given a game $\mathcal{G}$ with $|N|$ players and $|\mathrm{St}|$ states and budget β, it holds that*

$$|\mathcal{K}(\mathcal{G}, \beta)| = \frac{\beta + 1}{m} \binom{\beta + m}{\beta + 1},$$

with $m = |N \times \mathrm{St}|$ being the number of pairs of possible agents and states.

From Proposition 5 we derive that the number of possible subsidy schemes is *polynomial* in the budget β and singly *exponential* in both the number of agents and states in the game. At this point, solving Weak Implementation can be done with the following procedure:

1. Guess:
   - a subsidy scheme $\kappa \in \mathcal{K}(\mathcal{G}, \beta)$,
   - a state $s \in \mathrm{St}$ for every player $i \in \mathrm{N}$, and
   - punishment memoryless strategies $(\vec{\sigma}_{-1}, \ldots, \vec{\sigma}_{-n})$ for all players $i \in \mathrm{N}$;
2. Compute $(\mathcal{G}, \kappa)$;
3. Compute $z \in \mathbb{R}^{\mathrm{N}}$;
4. Compute the game $(\mathcal{G}, \kappa)[z]$ by removing the states s such that $\mathtt{pun}_i(s) \leq z_i$ for some player i and the transitions $(s, \vec{\mathsf{a}}_{-i})$ that are not z_i secure for player i;
5. Check whether there exists an ultimately periodic path π in $(\mathcal{G}, \kappa)[z]$ such that $\pi \models \varphi$ and $z_i \leq \mathsf{pay}_i(\pi)$ for every player $i \in \mathrm{N}$.

Since the set $\mathcal{K}(\mathcal{G}, \beta)$ is finitely bounded (Proposition 5), and punishment strategies only need to be memoryless, thus also finitely bounded, clearly step 1 can be guessed nondeterministically. Moreover, each of the guessed elements is of polynomial size, thus this step can be done (deterministically) in polynomial space. Step 2 clearly can be done in polynomial time. Step 3 can also be done in polynomial time since, given $(\vec{\sigma}_{-1}, \ldots, \vec{\sigma}_{-n})$, we can compute z solving $|\mathrm{N}|$ one-player mean-payoff games, one for each player i [35, Thm. 6]. For step 5, we will use Theorem 4 and consider two cases, one for LTL specifications and one for $\mathsf{GR}(1)$ specifications. Firstly, for LTL specifications, consider the formula

$$\varphi_{\mathrm{WI}} := \varphi \wedge \bigwedge_{i \in \mathrm{N}} (\mathsf{mp}(i) \geq z_i)$$

written in $\mathsf{LTL}^{\mathsf{Lim}}$ [7], an extension of LTL where statements about mean-payoff values over a given weighted arena can be made.[2] The semantics of the temporal operators of $\mathsf{LTL}^{\mathsf{Lim}}$ is just like the one for LTL over infinite computation paths $\pi = s_0, s_1, s_3 \ldots$. On the other hand, the meaning of $\mathsf{mp}(i) \geq z_i$ is simply that such an atomic formula is true if, and only if, the mean-payoff value of π with respect to player i is greater or equal to z_i, a constant real value; that is, $\mathsf{mp}(i) \geq z_i$ is true in π if and only if $\mathsf{pay}_i(\pi) = \mathsf{mp}(\mathsf{w}_i(\pi))$ is greater or equal than constant value z_i. Formula φ_{WI} corresponds exactly to $2(b)$ in Theorem 4. Furthermore, since every path in $(\mathcal{G}, \kappa)[z]$ satisfies condition $2(a)$ of Theorem 4, every computation path of $(\mathcal{G}, \kappa)[z]$ that satisfies φ_{WI} is a witness to the Weak Implementation problem.

▶ **Theorem 6.** Weak Implementation *with* LTL *specifications is* PSPACE*-complete.*

Proof. Membership follows from the procedure above and the fact that model checking for $\mathsf{LTL}^{\mathsf{Lim}}$ is PSPACE-complete [7]. Hardness follows from the fact that LTL model checking is a special case of Weak Implementation. For instance, consider the case in which all weights for all players are set to the same value, say 0, and the principal has budget $\beta = 0$. ◀

Case with GR(1) specifications. One of the main bottlenecks of our procedure to solve Weak Implementation lies in step 5, where we solve an $\mathsf{LTL}^{\mathsf{Lim}}$ model checking problem. To reduce the complexity of our decision procedure, we consider Weak Implementation with the specification φ expressed in the $\mathsf{GR}(1)$ sublanguage of LTL. With this specification language, the path finding problem can be solved without model-checking the $\mathsf{LTL}^{\mathsf{Lim}}$ formula given before. In order to do this, we can define a linear program (LP) such that the LP has a solution if and only if $\mathrm{WI}(\mathcal{G}, \varphi, \beta) \neq \varnothing$. From our previous procedure, observe that step 1 can be done nondeterministically in polynomial time, and steps 2–4 can be done

[2] The formal semantics of $\mathsf{LTL}^{\mathsf{Lim}}$ can be found in [7]. We prefer to give only an informal description here.

(deterministically) in polynomial time. Furthermore, using LP, we also can check step 5 deterministically in polynomial time. For the lower-bound, we use [33] and note that if $\varphi = \top$ and $\beta = 0$, then the problem reduces to checking whether the underlying mp game has a Nash equilibrium. Based on the above observations, we have the following result.

▶ **Theorem 7.** Weak Implementation *with* GR(1) *specifications is* NP*-complete.*

Proof sketch. For the upper bound, we define an LP of size polynomial in $(\mathcal{G}, \kappa)$ having a solution if and only if there is an ultimately periodic path π such that $z_i \leq \mathsf{pay}_i(\pi)$ and satisfies the GR(1) specification. Recall that φ has the following form

$$\varphi = \bigwedge_{l=1}^{m} \mathbf{GF}\psi_l \to \bigwedge_{r=1}^{n} \mathbf{GF}\theta_r,$$

and let $V(\psi_l)$ and $V(\theta_r)$ be the subset of states in $(\mathcal{G}, \kappa)$ that satisfy the Boolean combinations ψ_l and θ_r, respectively. Property φ is satisfied on π if, and only if, either π visits every state in $V(\theta_r)$ infinitely often or some of the states in $V(\psi_l)$ only a finite number of times. For the game $(\mathcal{G}, \kappa)[z]$, let $W = (V, E, (\mathsf{w}_a)_{a \in \mathrm{N}})$ be the underlying multi-weighted graph, and for every edge $e \in E$ introduce a variable x_e. Informally, the value of x_e is the number of times that e is used on a cycle. Formally, let $\mathsf{src}(e) = \{v \in V : \exists w\, e = (v, w) \in E\}$; $\mathsf{trg}(e) = \{v \in V : \exists w\, e = (w, v) \in E\}$; $\mathsf{out}(v) = \{e \in E : \mathsf{src}(e) = v\}$; and $\mathsf{in}(v) = \{e \in E : \mathsf{trg}(e) = v\}$. Now, consider ψ_l for some $1 \leq l \leq m$, and define the following linear program $\mathsf{LP}(\psi_l)$:

Eq1: $x_e \geq 0$ for each edge e – a basic consistency criterion;

Eq2: $\Sigma_{e \in E} x_e \geq 1$ – at least one edge is chosen;

Eq3: for each $a \in \mathrm{N}$, $\Sigma_{e \in E} \mathsf{w}_a(\mathsf{src}(e)) x_e \geq 0$ – total sum of any solution is non-negative;

Eq4: $\Sigma_{\mathsf{src}(e) \cap V(\psi_l) \neq \emptyset} x_e = 0$ – no state in $V(\psi_l)$ is in the cycle associated with the solution;

Eq5: for each $v \in V$, $\Sigma_{e \in \mathsf{out}(v)} x_e = \Sigma_{e \in \mathsf{in}(v)} x_e$ – this condition says that the number of times one enters a vertex is equal to the number of times one leaves that vertex.

$\mathsf{LP}(\psi_l)$ has a solution if and only if there is a path π in $\mathcal{G}$ such that $z_i \leq \mathsf{pay}_i(\pi)$ for every player i and visits $V(\psi_l)$ only *finitely many times*. Consider now the linear program $\mathsf{LP}(\theta_1, \ldots, \theta_n)$ defined as follows. Eq1–Eq3 as well as Eq5 are as in $\mathsf{LP}(\psi_l)$, and:

Eq4: for all $1 \leq r \leq n$, $\Sigma_{\mathsf{src}(e) \cap V(\theta_r) \neq \emptyset} x_e \geq 1$ – this condition says that, for every $V(\theta_r)$, at least one state in $V(\theta_r)$ is in the cycle associated with the solution of the linear program.

In this case, $\mathsf{LP}(\theta_1, \ldots, \theta_n)$ has a solution if and only if there exists a path π such that $z_i \leq \mathsf{pay}_i(\pi)$ for every player i and visits every $V(\theta_r)$ *infinitely many times*. Since the constructions above are polynomial in the size of both $(\mathcal{G}, \kappa)$ and φ, we can conclude it is possible to check in NP the statement that there is a path π satisfying φ such that $z_i \leq \mathsf{pay}_i(\pi)$ for every player i in the game if and only if one of the two linear programs defined above has a solution. For the lower-bound, we use [33] as discussed before. ◀

We now turn our attention to the strong implementation of the equilibrium design problem. As in this section, we first consider LTL specifications and then GR(1) specifications.

5 Equilibrium Design: Strong Implementation

Although the principal may find $\mathrm{WI}(\mathcal{G}, \varphi, \beta) \neq \varnothing$ to be good news, it might not be good enough. It could be that even though there is a desirable Nash equilibrium, the others might be undesirable. This motivates us to consider the *strong implementation* variant of equilibrium design. Intuitively, in a strong implementation, we require that *every* Nash equilibrium outcome satisfies the specification φ, for a *non-empty* set of outcomes. Then, let $\mathrm{SI}(\mathcal{G}, \varphi, \beta)$ denote the set of subsidy schemes κ given budget β over $\mathcal{G}$ such that:

1. $(\mathcal{G}, \kappa)$ has at least one Nash equilibrium outcome,
2. every Nash equilibrium outcome of $(\mathcal{G}, \kappa)$ satisfies φ.

Formally we define it as follows:

$$\mathrm{SI}(\mathcal{G}, \varphi, \beta) = \{\kappa \in \mathcal{K}(\mathcal{G}, \beta) : \mathrm{NE}(\mathcal{G}, \kappa) \neq \varnothing \wedge \forall \vec{\sigma} \in \mathrm{NE}(\mathcal{G}, \kappa) \text{ s.t. } \pi(\vec{\sigma}) \models \varphi\}.$$

This gives us the following decision problem:

▶ **Definition 8** (Strong Implementation). *Given a game $\mathcal{G}$, formula φ, and budget β:*

Is it the case that $\mathrm{SI}(\mathcal{G}, \varphi, \beta) \neq \varnothing$*?*

Strong Implementation can be solved with a 5-step procedure where the first four steps are as in Weak Implementation, and the last step (step 5) is as follows:

5. Check whether:

(a) there is no ultimately periodic path π in $(\mathcal{G}, \kappa)[z]$ such that $z_i \leq \mathsf{pay}_i(\pi)$ for each $i \in \mathrm{N}$;

(b) there is an ultimately periodic path π in $(\mathcal{G}, \kappa)[z]$ such that $\pi \models \neg\varphi$ and $z_i \leq \mathsf{pay}_i(\pi)$, for each $i \in \mathrm{N}$.

For step 5, observe that a positive answer to 5(a) or 5(b) is a counterexample to $\kappa \in \mathrm{SI}(\mathcal{G}, \varphi, \beta)$. Then, to carry out this procedure for the Strong Implementation problem with LTL specifications, consider the following $\mathsf{LTL}^{\mathsf{Lim}}$ formulae:

$$\varphi_{\exists} = \bigwedge_{i \in \mathrm{N}} (\mathsf{mp}(i) \geq z_i);$$
$$\varphi_{\forall} = \varphi_{\exists} \to \varphi.$$

Notice that the expression $\mathrm{NE}(\mathcal{G}, \kappa) \neq \varnothing$ can be expressed as "there exists a path π in $\mathcal{G}$ that satisfies formula $\varphi_{\exists}$". On the other hand, the expression $\forall \vec{\sigma} \in \mathrm{NE}(\mathcal{G}, \kappa)$ such that $\pi(\vec{\sigma}) \models \varphi$ can be expressed as "for every path π in $\mathcal{G}$, if π satisfies formula $\varphi_{\exists}$, then π also satisfies formula φ". Thus, using these two formulae, we obtain the following result.

▶ **Corollary 9.** Strong Implementation *with* LTL *specifications is* PSPACE*-complete.*

Proof. Membership follows from the fact that step 5(a) can be solved by existential $\mathsf{LTL}^{\mathsf{Lim}}$ model checking, whereas step 5(b) by universal $\mathsf{LTL}^{\mathsf{Lim}}$ model checking – both clearly in PSPACE by Savitch's theorem. Hardness is similar to the construction in Theorem 6. ◀

Case with GR(1) specifications. Notice that the first part, *i.e.*, $\mathrm{NE}(\mathcal{G}, \kappa) \neq \varnothing$ can be solved in NP [33]. For the second part, observe that

$$\forall \vec{\sigma} \in \mathrm{NE}(\mathcal{G}, \kappa) \text{ such that } \pi(\vec{\sigma}) \models \varphi$$

is equivalent to

$$\neg \exists \vec{\sigma} \in \mathrm{NE}(\mathcal{G}, \kappa) \text{ such that } \pi(\vec{\sigma}) \models \neg\varphi.$$

Thus we have

$$\neg\varphi = \bigwedge_{l=1}^{m} \mathbf{GF}\psi_l \wedge \neg\Big(\bigwedge_{r=1}^{n} \mathbf{GF}\theta_r\Big).$$

To check this, we modify the LP in Theorem 7. Specifically, we modify Eq4 in $\mathsf{LP}(\theta_1, \dots, \theta_n)$ to encode the θ-part of $\neg\varphi$. Thus, we have the following equation in $\mathsf{LP}'(\theta_1, \dots, \theta_n)$:

Eq4: there exists r, $1 \leq r \leq n$, $\Sigma_{\mathsf{src}(e) \cap V(\theta_r) \neq \emptyset} x_e = 0$ – this condition ensures that at least one set $V(\theta_r)$ does not have any state in the cycle associated with the solution.

In this case, $\mathsf{LP}'(\theta_1, \dots, \theta_n)$ has a solution if and only if there is a path π such that $z_i \leq \mathsf{pay}_i(\pi)$ for every player i and, for at least one $V(\theta_r)$, its states are visited only *finitely many times*. Thus, we have a procedure that checks if there is a path π that satisfies $\neg\varphi$ such that $z_i \leq \mathsf{pay}_i(\pi)$ for every player i, if and only if both linear programs have a solution. Using this new construction, we can now prove the following result.

▶ **Theorem 10.** Strong Implementation *with* $\mathsf{GR}(1)$ *specifications is* Σ_2^{P}*-complete.*

Proof sketch. For membership, observe that by rearranging the problem statement, we have the following question: Check whether the following expression is true

$$\exists \kappa \in \mathcal{K}(\mathcal{G}, \beta), \tag{1}$$

$$\exists \vec{\sigma} \in \sigma_1 \times \cdots \times \sigma_n, \text{ such that } \vec{\sigma} \in \mathrm{NE}(\mathcal{G}, \kappa), \tag{2}$$

and

$$\forall \vec{\sigma}' \in \sigma_1 \times \cdots \times \sigma_n, \text{ if } \vec{\sigma}' \in \mathrm{NE}(\mathcal{G}, \kappa) \text{ then } \pi(\vec{\sigma}') \models \varphi. \tag{3}$$

Statement (2) can be checked in NP (Theorem 4), whereas verifying statement (3) is in coNP; to see this, notice that we can rephrase (3) as follows: $\neg\exists z \in \{\mathtt{pun}_i(s) : s \in \mathrm{St}\}^{\mathrm{N}}$ such that both $\mathsf{LP}(\psi_l)$ and $\mathsf{LP}'(\theta_1, \dots, \theta_n)$ have a solution in $(\mathcal{G}, \kappa)[z]$. Thus, membership in Σ_2^{P} follows. We prove hardness via a reduction from QSAT_2 (satisfiability of quantified Boolean formulae with 2 alternations), which is known to be Σ_2^{P}-complete [28]. ◀

6 Optimality and Uniqueness of Solutions

Having asked the questions studied in the previous sections, the principal – the *designer* in the equilibrium design problem – may want to explore further information. Because the power of the principal is limited by its budget, and because from the point of view of the system, it may be associated with a reward (*e.g.*, money, savings, etc.) or with the inverse of the amount of a finite resource (*e.g.*, time, energy, etc.) an obvious question is asking about *optimal* solutions. This leads us to *optimisation* variations of the problems we have studied. Informally, in this case, we ask what is the least budget that the principal needs to ensure that the implementation problems have positive solutions. The principal may also want to know whether a given subsidy scheme is *unique*, so that there is no point in looking for any other solutions to the problem. In this section, we investigate these kind of problems, and classify our study into two parts, one corresponding to the Weak Implementation problem and another one corresponding to the Strong Implementation problem.

6.1 Optimality and Uniqueness in the Weak Domain

We can now define formally some of the problems that we will study in the rest of this section. To start, the optimisation variant for Weak Implementation is defined as follows.

▶ **Definition 11** (Opt-WI). *Given a game* $\mathcal{G}$ *and a specification formula* φ*:*

What is the optimum budget β *such that* $\mathrm{WI}(\mathcal{G}, \varphi, \beta) \neq \varnothing$*?*

Another natural problem, which is related to Opt-WI, is the "exact" variant – a membership question. In this case, in addition to $\mathcal{G}$ and φ, we are also given an integer b, and ask whether it is indeed the smallest amount of budget that the principal has to spend for some optimal weak implementation. This decision problem is formally defined as follows.

▶ **Definition 12** (Exact-WI). *Given a game $\mathcal{G}$, a specification formula φ, and an integer b:*

Is b equal to the optimum budget for $\mathrm{WI}(\mathcal{G}, \varphi, \beta) \neq \varnothing$*?*

To study these problems, it is useful to introduce some concepts first. More specifically, let us introduce the concept of *implementation efficiency.* We say that a Weak Implementation (resp. Strong Implementation) is *efficient* if $\beta = \mathsf{cost}(\kappa)$ and there is no κ' such that $\mathsf{cost}(\kappa') < \mathsf{cost}(\kappa)$ and $\kappa' \in \mathrm{WI}(\mathcal{G}, \varphi, \beta)$ (resp. $\kappa' \in \mathrm{SI}(\mathcal{G}, \varphi, \beta)$). In addition to the concept of efficiency for an implementation problem, it is also useful to have the following result.

▶ **Proposition 13.** *Let z_i be the largest payoff that player i can get after deviating from a path π. The optimum budget is an integer between 0 and $\sum_{i \in \mathrm{N}} z_i \cdot (|\mathrm{St}| - 1)$.*

Using Proposition 13, we can show that both Opt-WI and Exact-WI can be solved in PSPACE for LTL specifications. Intuitively, the reason is that we can use the upper bound given by Proposition 13 to go through all possible solutions in exponential time, but using only nondeterministic polynomial space. Formally, we have the following results.

▶ **Theorem 14.** Opt-WI *with* LTL *specifications is* FPSPACE*-complete.*

▶ **Corollary 15.** Exact-WI *with* LTL *specifications is* PSPACE*-complete.*

The fact that both Opt-WI and Exact-WI with LTL specifications can be answered in, respectively, FPSPACE and PSPACE does not come as a big surprise: checking an instance can be done using polynomial space and there are only exponentially many instances to be checked. However, for Opt-WI and Exact-WI with GR(1) specifications, these two problems are more interesting.

▶ **Theorem 16.** Opt-WI *with* GR(1) *specifications is* $\mathsf{FP}^{\mathsf{NP}}$*-complete.*

Proof sketch. Membership follows from the fact that the search space, which is bounded as in Proposition 13, can be explored using binary search and Weak Implementation as an oracle. More precisely, we can find the smallest budget β such that $\mathrm{WI}(\mathcal{G}, \varphi, \beta) \neq \varnothing$ by checking every possible value for β, which lies between 0 and 2^n, where n is the length of the encoding of the instance. Since, due to the binary search routine, we need logarithmically many calls to the NP oracle (*i.e.*, to Weak Implementation), in the end we have a searching procedure that would run in polynomial time. For the lower bound, we reduce from TSP Cost (the optimal travelling salesman problem), which is $\mathsf{FP}^{\mathsf{NP}}$-complete [28]. ◀

▶ **Corollary 17.** Exact-WI *with* GR(1) *specifications is* D^{P}*-complete.*

Proof. For membership, observe that an input is a "yes" instance of Exact-WI if and only if it is a "yes" instance of Weak Implementation *and* a "yes" instance of Weak Implementation Complement (the problem where one asks whether $\mathrm{WI}(\mathcal{G}, \varphi, \beta) = \varnothing$). Since the former problem is in NP and the latter problem is in coNP, membership in D^{P} follows. For the lower bound, we use the same reduction technique as in Theorem 16, and reduce from Exact TSP, a problem known to be D^{P}-hard [28, 29]. ◀

Following [27], we may naturally ask whether the optimal solution given by Opt-WI is unique. We call this problem UOpt-WI. For some fixed budget β, it may be the case that for two subsidy schemes $\kappa, \kappa' \in \mathrm{WI}(\mathcal{G}, \varphi, \beta)$ – we assume the implementation is efficient – we have $\kappa \neq \kappa'$ and $\mathsf{cost}(\kappa) = \mathsf{cost}(\kappa')$. With LTL specifications, it is not difficult to see that we can solve UOpt-WI in polynomial space. Therefore, we have the following result.

▶ **Corollary 18.** UOpt-WI *with* LTL *specifications is* PSPACE*-complete.*

For GR(1) specifications, we reason about UOpt-WI using the following procedure:

1. Find the exact budget using binary search and Weak Implementation as an oracle;
2. Use an NP oracle once to guess two distinct subsidy schemes with precisely this budget; if no such subsidy schemes exist, return "yes"; otherwise, return "no".

The above decision procedure clearly is in Δ_2^{P} (for the upper bound). Furthermore, since Theorem 16 implies Δ_2^{P}-hardness [22] (for the lower bound), we have the following corollary.

▶ **Corollary 19.** UOpt-WI *with* GR(1) *specifications is* Δ_2^{P}*-complete.*

6.2 Optimality and Uniqueness in the Strong Domain

In this subsection, we study the same problems as in the previous subsection but with respect to the Strong Implementation variant of the equilibrium design problem. We first formally define the problems of interest and then present the two first results.

▶ **Definition 20** (Opt-SI). *Given a game $\mathcal{G}$ and a specification formula φ:*

$$\textit{What is the optimum budget } \beta \textit{ such that } \mathrm{SI}(\mathcal{G},\varphi,\beta) \neq \varnothing\textit{?}$$

▶ **Definition 21** (Exact-SI). *Given a game $\mathcal{G}$, a specification formula φ, and an integer b:*

$$\textit{Is } b \textit{ equal to the optimum budget for } \mathrm{SI}(\mathcal{G},\varphi,\beta) \neq \varnothing\textit{?}$$

For the same reasons discussed in the weak versions of these two problems, we can prove the following two results with respect to games with LTL specifications.

▶ **Theorem 22.** Opt-SI *with* LTL *specifications is* FPSPACE*-complete.*

▶ **Corollary 23.** Exact-SI *with* LTL *specifications is* PSPACE*-complete.*

For GR(1) specifications, observe that using the same arguments for the upper-bound of Opt-WI with GR(1) specifications, we obtain the upper-bound for Opt-SI with GR(1) specifications. Then, it follows that Opt-SI is in $\mathsf{FP}^{\Sigma_2^{\mathsf{P}}}$. For hardness, we define an $\mathsf{FP}^{\Sigma_2^{\mathsf{P}}}$-complete problem, namely Weighted MinQSAT$_2$. Recall that in QSAT$_2$ we are given a Boolean 3DNF formula $\psi(\mathbf{x},\mathbf{y})$ and sets $\mathbf{x} = \{x_1,\ldots,x_n\}, \mathbf{y} = \{y_1,\ldots,y_m\}$, with a set of terms $T = \{t_1,\ldots,t_k\}$. Define Weighted MinQSAT$_2$ as follows. Given $\psi(\mathbf{x},\mathbf{y})$ and a weight function $\mathsf{c} : \mathbf{x} \to \mathbb{Z}^{\geq}$, Weighted MinQSAT$_2$ is the problem of finding an assignment $\vec{\mathbf{x}} \in \{0,1\}^n$ with the least total weight such that $\psi(\mathbf{x},\mathbf{y})$ is true for every $\vec{\mathbf{y}} \in \{0,1\}^m$. Observe that Weighted MinQSAT$_2$ generalises MinQSAT$_2$, which is known to be $\mathsf{FP}^{\Sigma_2^{\mathsf{P}}[\log n]}$-hard [12], *i.e.*, MinQSAT$_2$ is an instance of Weighted MinQSAT$_2$, where all weights are 1.

▶ **Theorem 24.** Weighted MinQSAT$_2$ *is* $\mathsf{FP}^{\Sigma_2^{\mathsf{P}}}$*-complete.*

Proof. Membership follows from the upper-bound of MinQSAT$_2$ [12]: since we have an exponentially large input with respect to that of MinQSAT$_2$, by using binary search we will need polynomially many calls to the Σ_2^{P} oracle. Hardness is immediate [12]. ◀

Now that we have an $\mathsf{FP}^{\Sigma_2^{\mathsf{P}}}$-hard problem in our hands, we can proceed to determine the complexity class of Opt-SI with GR(1) specifications. For the upper bound we one can use arguments analogous to those in Theorem 16. For the lower bound, one can reduce from Weighted MinQSAT$_2$. Formally, we have:

▶ **Theorem 25.** OPT-SI *with* GR(1) *specifications is* $\mathsf{FP}^{\Sigma_2^{\mathsf{P}}}$*-complete.*

▶ **Corollary 26.** EXACT-SI *with* GR(1) *specifications is* $\mathsf{D}_2^{\mathsf{P}}$*-complete.*

Proof. Membership follows from the fact that an input is a "yes" instance of EXACT-SI (with GR(1) specifications) if and only if it is a "yes" instance of STRONG IMPLEMENTATION *and* a "yes" instance of STRONG IMPLEMENTATION COMPLEMENT, the decision problem where we ask $\mathrm{SI}(\mathcal{G}, \varphi, \beta) = \varnothing$ instead. The lower bound follows from the hardness of STRONG IMPLEMENTATION and STRONG IMPLEMENTATION COMPLEMENT problems, which immediately implies $\mathsf{D}_2^{\mathsf{P}}$-hardness [1, Lemma 3.2]. ◀

Furthermore, analogous to UOPT-WI, we also have the following corollaries.

▶ **Corollary 27.** UOPT-SI *with* LTL *specifications is* PSPACE*-complete.*

▶ **Corollary 28.** UOPT-SI *with* GR(1) *specifications is* Δ_3^{P}*-complete.*

7 Conclusions & Related and Future Work

Equilibrium design vs. mechanism design – connections with Economic theory

Although equilibrium design is closely related to mechanism design, as typically studied in game theory [21], the two are not exactly the same. Two key features in mechanism design are the following. Firstly, in a mechanism design problem, the designer is not given a game structure, but instead is asked to provide one; in that sense, a mechanism design problem is closer to a rational synthesis problem [14, 16]. Secondly, in a mechanism design problem, the designer is only interested in the game's outcome, which is given by the payoffs of the players in the game; however, in equilibrium design, while the designer is interested in the payoffs of the players as these may need to be perturbed by its budget, the designer is also interested – and in fact primarily interested – in the satisfaction of a temporal logic goal specification, which the players in the game do not take into consideration when choosing their individual rational choices; in that sense, equilibrium design is closer to rational verification [17] than to mechanism design. Thus, equilibrium design is a new computational problem that sits somewhere in the middle between mechanism design and rational verification/synthesis. Technically, in equilibrium design we go beyond rational synthesis and verification through the additional design of subsidy schemes for incentivising behaviours in a concurrent and multi-agent system, but we do not require such subsidy schemes to be incentive compatible mechanisms, as in mechanism design theory, since the principal may want to reward only a group of players in the game so that its temporal logic goal is satisfied, while rewarding other players in the game in an unfair way – thus, leading to a game with a suboptimal social welfare measure. In this sense, equilibrium design falls short with respect to the more demanding social welfare requirements often found in mechanism design theory.

Equilibrium design vs. rational verification – connections with Computer science

Typically, in rational synthesis and verification [14, 16, 17, 23] we want to check whether a property is satisfied on some/every Nash equilibrium computation run of a reactive, concurrent, and multi-agent system. These verification problems are primarily concerned with qualitative properties of a system, while assuming rationality of system components. However, little attention is paid to quantitative properties of the system. This drawback has been recently identified and some work has been done to cope with questions where both qualitative and quantitative concerns are considered [3, 6, 9, 10, 11, 18, 20, 34]. Equilibrium design is new and different approach where this is also the case. More specifically, as in a mechanism design problem, through the introduction of an external principal – the designer in the equilibrium design problem – we can account for overall qualitative properties of a system (the principal's goal given by an LTL or a GR(1) specification) as well as for quantitative concerns (optimality of solutions constrained by the budget to allocate additional rewards/resources). Our framework also mixes qualitative and quantitative features in a different way: while system components are only interested in maximising a quantitative payoff, the designer is primarily concerned about the satisfaction of a qualitative (logic) property of the system, and only secondarily about doing it in a quantitatively optimal way.

Equilibrium design vs. repair games and normative systems – connections with AI

In recent years, there has been an interest in the analysis of rational outcomes of multi-agent systems modelled as multi-player games. This has been done both with modelling and with verification purposes. In those multi-agent settings, where AI agents can be represented as players in a multi-player game, a focus of interest is on the analysis of (Nash) equilibria in such games [8, 17]. However, it is often the case that the existence of Nash equilibria in a multi-player game with temporal logic goals may not be guaranteed [16, 17]. For this reason, there has been already some work on the introduction of desirable Nash equilibria in multi-player games [2, 30]. This problem has been studied as a repair problem [2] in which either the preferences of the players (given by winning conditions) or the actions available in the game are modified; the latter one also being achieved with the use of normative systems [30]. In equilibrium design, we do not directly modify the preferences of agents in the system, since we do not alter their goals or choices in the game, but we indirectly influence their rational behaviour by incentivising players to visit, or to avoid, certain states of the overall system. We studied how to do this in an (individually) optimal way with respect to the preferences of the principal in the equilibrium design problem. However, this may not always be possible, for instance, because the principal's temporal logic specification goal is just not achievable, or because of constraints given by its limited budget.

Future work: social welfare requirements and practical implementation

As discussed before, a key difference with mechanism design is that social welfare requirements are not considered [25]. However, a benevolent principal might not see optimality as an individual concern, and instead consider the welfare of the players in the design of a subsidy scheme. In that case, concepts such as the *utilitarian social welfare* may be undesirable as the social welfare maximising the payoff received by players might allocate all the budget to only one player, and none to the others. A potentially better option is to improve fairness in the allocation of the budget by maximising the *egalitarian social welfare*. Finally, given that the complexity of equilibrium design is much better than that of rational synthesis/verification, we should be able to have efficient implementations, for instance, as an extension of EVE [19].

References

1 Gadi Aleksandrowicz, Hana Chockler, Joseph Y. Halpern, and Alexander Ivrii. The Computational Complexity of Structure-based Causality. *J. Artif. Int. Res.*, 58(1):431–451, January 2017. URL: `http://dl.acm.org/citation.cfm?id=3176764.3176775`.

2 S. Almagor, G. Avni, and O. Kupferman. Repairing Multi-Player Games. In *CONCUR*, volume 42 of *LIPIcs*, pages 325–339. Schloss Dagstuhl, 2015.

3 S. Almagor, O. Kupferman, and G. Perelli. Synthesis of Controllable Nash Equilibria in Quantitative Objective Games. In *IJCAI*, pages 35–41, 2018.

4 B. Aminof, V. Malvone, A. Murano, and S. Rubin. Graded Strategy Logic: Reasoning about Uniqueness of Nash Equilibria. In *AAMAS*, pages 698–706. ACM, 2016.

5 R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of Reactive(1) designs. *Journal of Computer and System Sciences*, 78(3):911–938, 2012.

6 A. Bohy, V. Bruyère, E. Filiot, and J. Raskin. Synthesis from LTL Specifications with Mean-Payoff Objectives. In *TACAS*, pages 169–184, 2013.

7 U. Boker, K. Chatterjee, T. A. Henzinger, and O. Kupferman. Temporal Specifications with Accumulative Values. *ACM Transactions on Computational Logic*, 15(4):27:1–27:25, 2014. `doi:10.1145/2629686`.

8 P. Bouyer, R. Brenguier, N. Markey, and M. Ummels. Pure Nash Equilibria in Concurrent Deterministic Games. *Logical Methods in Computer Science*, 11(2), 2015.

9 K. Chatterjee and L. Doyen. Energy parity games. *Theoretical Computer Science*, 458:49–60, 2012.

10 K. Chatterjee, L. Doyen, T. Henzinger, and J. Raskin. Generalized Mean-payoff and Energy Games. In *FSTTCS*, pages 505–516, 2010. `doi:10.4230/LIPIcs.FSTTCS.2010.505`.

11 K. Chatterjee, T. A. Henzinger, and M. Jurdzinski. Mean-Payoff Parity Games. In *LICS*, pages 178–187. IEEE Computer Society, 2005.

12 H. Chockler and J. Halpern. Responsibility and Blame: A Structural-Model Approach. *Journal of Artificial Intelligence Research*, 22:93–115, 2004.

13 Aldric Degorre, Laurent Doyen, Raffaella Gentilini, Jean-François Raskin, and Szymon Toruńczyk. Energy and Mean-Payoff Games with Imperfect Information. In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic*, pages 260–274, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

14 D. Fisman, O. Kupferman, and Y. Lustig. Rational Synthesis. In *TACAS*, volume 6015 of *LNCS*, pages 190–204. Springer, 2010.

15 J. Gutierrez, P. Harrenstein, G. Perelli, and M. Wooldridge. Nash Equilibrium and Bisimulation Invariance. In *CONCUR*, volume 85 of *LIPIcs*, pages 17:1–17:16. Schloss Dagstuhl, 2017.

16 J. Gutierrez, P. Harrenstein, and M. Wooldridge. Iterated Boolean Games. *Information and Computation*, 242:53–79, 2015.

17 J. Gutierrez, P. Harrenstein, and M. Wooldridge. From Model Checking to Equilibrium Checking: Reactive Modules for Rational Verification. *Artificial Intelligence*, 248:123–157, 2017.

18 J. Gutierrez, A. Murano, G. Perelli, S. Rubin, and M. Wooldridge. Nash Equilibria in Concurrent Games with Lexicographic Preferences. In *IJCAI*, pages 1067–1073, 2017. `doi:10.24963/ijcai.2017/148`.

19 J. Gutierrez, M. Najib, G. Perelli, and M. Wooldridge. EVE: A Tool for Temporal Equilibrium Analysis. In *ATVA*, volume 11138 of *LNCS*, pages 551–557. Springer, 2018.

20 J. Gutierrez, M. Najib, G. Perelli, and M. Wooldridge. On Computational Tractability for Rational Verification. In *IJCAI*, 2019. To appear.

21 L. Hurwicz and S. Reiter. *Designing Economic Mechanisms.* Cambridge University Press, 2006.

22 M. Krentel. The Complexity of Optimization Problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988. `doi:10.1016/0022-0000(88)90039-6`.

23 O. Kupferman, G. Perelli, and M. Y. Vardi. Synthesis with Rational Environments. *Annals of Mathematics and Artificial Intelligence*, 78(1):3–20, 2016.

24 M. Wooldridge and U. Endriss and S. Kraus and J. Lang. Incentive engineering for Boolean games. *Artificial Intelligence*, 195:418–439, 2013. `doi:10.1016/j.artint.2012.11.003`.

25 M. Maschler, E. Solan, and S. Zamir. *Game Theory.* Cambridge University Press, 2013.

26 M.J. Osborne and A. Rubinstein. *A Course in Game Theory.* MIT Press, 1994.

27 C. Papadimitriou. On the Complexity of Unique Solutions. *Journal of the ACM*, 31(2):392–400, 1984. doi:10.1145/62.322435.

28 C. Papadimitriou. *Computational complexity*. Addison-Wesley, Reading, Massachusetts, 1994.

29 C. Papadimitriou and M. Yannakakis. The Complexity of Facets (and some Facets of Complexity). *Journal of Computer and System Sciences*, 28(2):244–259, 1984.

30 G. Perelli. Enforcing Equilibria in Multi-Agent Systems. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, pages 188–196, 2019. URL: http://dl.acm.org/citation.cfm?id=3306127.3331692.

31 A. Pnueli. The Temporal Logic of Programs. In *FOCS*, pages 46–57. IEEE, 1977.

32 A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *POPL*, pages 179–190. ACM Press, 1989.

33 M. Ummels and D. Wojtczak. The Complexity of Nash Equilibria in Limit-Average Games. In *CONCUR*, pages 482–496, 2011. doi:10.1007/978-3-642-23217-6_32.

34 Y. Velner, K. Chatterjee, L. Doyen, T. Henzinger, A. Rabinovich, and J. Raskin. The Complexity of Multi-Mean-Payoff and Multi-Energy Games. *Information and Computation*, 241:177–196, 2015.

35 U. Zwick and M. Paterson. The Complexity of Mean Payoff Games on Graphs. *Theoretical Computer Science*, 158(1):343–359, 1996. doi:10.1016/0304-3975(95)00188-3.

Partial Order Reduction for Reachability Games

Frederik Meyer Bønneland
Department of Computer Science, Aalborg University, Denmark
frederikb@cs.aau.dk

Peter Gjøl Jensen
Department of Computer Science, Aalborg University, Denmark
pgj@cs.aau.dk

Kim G. Larsen
Department of Computer Science, Aalborg University, Denmark
kgl@cs.aau.dk

Marco Muñiz
Department of Computer Science, Aalborg University, Denmark
muniz@cs.aau.dk

Jiří Srba
Department of Computer Science, Aalborg University, Denmark
srba@cs.aau.dk

Abstract

Partial order reductions have been successfully applied to model checking of concurrent systems and practical applications of the technique show nontrivial reduction in the size of the explored state space. We present a theory of partial order reduction based on stubborn sets in the game-theoretical setting of 2-player games with reachability/safety objectives. Our stubborn reduction allows us to prune the interleaving behaviour of both players in the game, and we formally prove its correctness on the class of games played on general labelled transition systems. We then instantiate the framework to the class of weighted Petri net games with inhibitor arcs and provide its efficient implementation in the model checker TAPAAL. Finally, we evaluate our stubborn reduction on several case studies and demonstrate its efficiency.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases Petri nets, games, synthesis, partial order reduction, stubborn sets

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.23

Funding The research leading to these results has received funding from the project DiCyPS funded by the Innovation Fund Denmark and the ERC Advanced Grant LASSO. The last author is partially affiliated with FI MU, Brno, Czech Republic.

1 Introduction

The state space explosion problem is the main obstacle for model checking of concurrent systems as even very simple processes running in parallel can produce an exponentially large number of possible interleavings and hence make the state space search practically intractable. One of the ways to tame this problem is by employing a variety of partial order reduction techniques, including the seminal work on stubborn set reductions by Valmari et al. [24, 23, 25].

As our main contribution, we generalize the theory of partial order reductions into the framework of 2-player games. The idea is that whenever one of the players can perform a series of moves in different sub-components of the system in parallel (without being disturbed by the other player), we may apply the classical stubborn set reductions in order to reduce the number of interleavings of independent actions. There is a number of subtle points that

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 23; pp. 23:1–23:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

one has to satisfy so that the reduced game preserves the winning strategies of both players. We formulate a number of sufficient conditions that define the notion of a *stable* stubborn set reduction that guarantees the preservation of winning strategies for both players in the game. In the setting of general game labelled transition systems, we formally prove the correctness of stable reductions and we demonstrate the applicability of the framework on weighted Petri net games with inhibitor arcs. We show how to approximate in a syntax-driven manner the conditions of a stable Petri net game reduction and provide an efficient, open source implementation in the model checker TAPAAL [6]. We also implement a game engine based on dependency graphs, following the approach from [14, 5], and on several case studies evaluate the game engine both with and without the use of the stable stubborn set reduction. The experiments demonstrate that the computation of the stubborn sets has only a minor overhead and has the potential of achieving exponential reduction both in the running time as well as in the number of searched configurations. We believe that this is the first implementation of 2-player game partial order reduction technique for Petri nets working in practice.

Related Work. Partial order reductions in the non-game setting for linear time properties have previously been studied [19, 23, 18, 17] which lends itself towards the safeness or liveness properties we want to preserve for winning states. In [19] and [23] Peled and Valmari present partial order reductions for general LTL. In [18] Lehmann et al. study stubborn sets applied to a subset of LTL properties called simple linear time properties which does not require all the requirements for general LTL preservation.

The extension of partial order reductions to game-oriented formalisms and verification tasks has not yet received much attention in the literature. In [10] partial order reductions for LTL without the next operator are adapted to a subset of alternating-time temporal logic and applied to multi-agent systems. The authors considers games with imperfect information, however they also show that their technique does not work for strategies with perfect information. We assume an antagonistic environment and focus on preserving the existence of winning strategies with perfect information, reducing the state space and improving existing controller synthesis algorithms. Partial order reduction for the problem of checking bisimulation equivalance between two labelled transition systems is presented in [9]. Our partial order reduction is applied directly to a labelled transition system while theirs are applied to the bisimulation game graph. While the setting is distinctly different, our approach is more general as we allow for mixed states, provide less information to the controller, and allow for reduction in both controllable as well as environmental states. Moreover, we provide an implementation of the on-the-fly strategy synthesis algorithm and argue by a number of case studies for its practical applicability.

The work on partial order reductions for modal mu-calculus and CTL (see e.g. [21, 26]) allows us in principle to encode the game semantics as a part of the mu-calculus formula, however, there is to the best of our knowledge no literature documenting the practical applicability of this approach.

Complexity and decidability results for control synthesis in Petri nets games are not encouraging. The control synthesis problem is for many instances of Petri net formalisms undecidable [1, 2], including those that allow for inhibition [2] which we utilise to model our case studies. If the problem is decidable for a given instance of a Petri net formalism (like e.g. for bounded nets) then it is usually of exponential complexity. In fact, most questions about the behaviour of bounded Petri nets are at least PSPACE-hard [11]. Among these questions is the existence of an infinite run [7] that we need to test as one of the sufficient conditions for applying stubborn set reductions to games. Instead of using exact infinite run detection approaches like in [7], we opt for efficient overapproximation algorithms to detect cycles using both syntactic and local state information.

2 Preliminaries

▶ **Definition 1** (Game Labelled Transition System). *A (deterministic) Game Labelled Transition System (GLTS) is a tuple $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ where $\mathcal{S}$ is a set of states, A_1 is a finite set of actions for player 1 (the controller), A_2 is a finite set of actions for player 2 (the environment) where $A_1 \cap A_2 = \emptyset$ and $A = A_1 \cup A_2$, $\rightarrow \subseteq \mathcal{S} \times A \times \mathcal{S}$ is a transition relation s.t. if $(s, a, s') \in \rightarrow$ and $(s, a, s'') \in \rightarrow$ then $s' = s''$, and $Goal \subseteq \mathcal{S}$ is a set of goal states.*

Let $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ be a fixed GLTS for the remainder of the section. Whenever $(s, a, s') \in \rightarrow$ we write $s \xrightarrow{a} s'$ and say that a is enabled in s and can be *fired* in s yielding s'. Otherwise we say that a is *disabled* in s. The set of *enabled* player i actions where $i \in \{1, 2\}$ in a state $s \in \mathcal{S}$ is given by $en_i(s) = \{a \in A_i \mid \exists s' \in \mathcal{S}.\ s \xrightarrow{a} s'\}$. The set of all enabled actions is given by $en(s) = en_1(s) \cup en_2(s)$. For a state $s \in \mathcal{S}$ where $en(s) \neq \emptyset$ if $en_2(s) = \emptyset$ then we call s a player 1 state, if $en_1(s) = \emptyset$ then we call s a player 2 state, and otherwise we call it a mixed state. The GLTS G is called non-mixed if all states are either player 1 or player 2 states. For a sequence of actions $w = a_1 a_2 \cdots a_n \in A^*$ we write $s \xrightarrow{w} s'$ if $s \xrightarrow{a_1} s_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} s'$. If $w \in A^\omega$, i.e. it is infinite, then we write $s \xrightarrow{w}$. Actions that are a part of w are said to occur in w. A sequence of states induced by $w \in A^* \cup A^\omega$ is called a *run* and is written as $\pi = s_0 s_1 \cdots$. We use $\Pi_G(s)$ to denote the set of all runs starting from a state $s \in \mathcal{S}$ in GLTS G, s.t. for all $s_0 s_1 \cdots \in \Pi_G(s)$ we have $s_0 = s$, and $\Pi_G = \bigcup_{s \in \mathcal{S}} \Pi_G(s)$ as the set of all runs. The length of a run π (number of states in the run) is given by the function $\ell : \Pi_G \rightarrow \mathbb{N}^0 \cup \{\infty\}$. A position in a run $\pi = s_0 s_1 \ldots \in \Pi_G(s)$ is a natural number $i \in \mathbb{N}^0$ that refers to the state s_i and is written as π_i. A position i can range from 0 to $\ell(\pi)$ s.t. if π is infinite then $i \in \mathbb{N}^0$ and otherwise $0 \leq i \leq \ell(\pi)$. Let $\Pi_G^{max}(s)$ be the set of all maximal runs starting from s, defined as $\Pi_G^{max}(s) = \{\pi \in \Pi_G(s) \mid \ell(\pi) = \infty \vee en(\pi_{\ell(\pi)}) = \emptyset)\}$. We omit the GLTS G from the subscript of run sets if it is clear from the context.

A reduced game is defined by a function called a reduction.

▶ **Definition 2** (Reduction). *Let $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ be a GLTS. A* reduction *is a function $St : \mathcal{S} \rightarrow 2^A$.*

▶ **Definition 3** (Reduced Game). *Let $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ be a GLTS and St be a reduction. The* reduced game *of G by the reduction St is given by $G_{St} = (\mathcal{S}, A_1, A_2, \underset{St}{\longrightarrow}, Goal)$ where $s \xrightarrow[St]{a} s'$ iff $s \xrightarrow{a} s'$ and $a \in St(s)$.*

The set of actions $St(s)$ is the *stubborn set* of s with the reduction St. The set of non-stubborn actions for s is defined as $\overline{St(s)} = A \setminus St(s)$.

A (memoryless) strategy is a function which proposes the next action player 1 wants to be fired.

▶ **Definition 4** (Strategy). *Let $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ be a GLTS. A* strategy *is a function $\sigma : \mathcal{S} \rightarrow A_1 \cup \{\bot\}$ where for all $s \in \mathcal{S}$ we have if $en_1(s) \neq \emptyset$ then $\sigma(s) \in en_1(s)$ else $\sigma(s) = \bot$.*

The intuition is that in order to ensure progress, player 1 always has to propose an action if she has an enabled action. Let σ be a fixed strategy for the remainder of the section. We define a function $next_\sigma(s)$ that returns the set of actions considered at $s \in \mathcal{S}$ under σ as:

$$next_\sigma(s) = \begin{cases} en_2(s) \cup \sigma(s) & \text{if } \sigma(s) \neq \bot \\ en_2(s) & \text{otherwise.} \end{cases}$$

Let $\Pi_\sigma^{max}(s) \subseteq \Pi^{max}(s)$ be the set of maximal runs subject to σ starting at $s \in \mathcal{S}$, defined as:

$$\Pi_\sigma^{max}(s) = \{\pi \in \Pi^{max}(s) \mid \forall i \in \{1, ..., \ell(\pi)\}.\ \exists a \in next_\sigma(\pi_{i-1}).\ \pi_{i-1} \xrightarrow{a} \pi_i)\}\ .$$

▶ **Definition 5** (Winning Strategy). *Let $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ be a GLTS and $s \in \mathcal{S}$ be a state. A strategy σ is a* winning strategy *for player 1 at s in G iff for all $\pi \in \Pi_\sigma^{max}(s)$ there exists a position i s.t. $\pi_i \in Goal$.*

If a state is winning for player 1 in G then no matter what action sequence the environment chooses, eventually a goal state is reached. Furthermore, for a given winning strategy σ at s in G there is a finite number $n \in \mathbb{N}$ such that a goal state is always reached with at most n action firings. We call the minimum such number the *strategy depth* of σ.

▶ **Definition 6** (Strategy Depth). *Let $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ be a GLTS, $s \in \mathcal{S}$ a winning state for player 1 in G where $s \notin Goal$, and σ a winning strategy at s in G. Then $n \in \mathbb{N}^0$ is the* depth *of σ at s in G if:*

- *for all $\pi \in \Pi_{G,\sigma}^{max}(s)$ there exists $0 \leq i \leq n$ s.t. $\pi_i \in Goal$, and*
- *there exists $\pi' \in \Pi_{G,\sigma}^{max}(s)$ s.t. $\pi'_n \in Goal$ and for all $0 \leq j < n$ we have $\pi'_j \notin Goal$.*

▶ **Lemma 7.** *Let $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ be a GLTS, $s \in \mathcal{S}$ a winning state for player 1 in G, and σ a winning strategy at s in G.*

1. *There exists $n \in \mathbb{N}^0$ that is the depth of σ at s in G.*
2. *For all $a \in next_\sigma(s)$ where $s \xrightarrow{a} s'$, the depth of σ at s' in G is m such that $0 \leq m < n$.*

A set of actions for a given state and a given set of goal states is called an interesting set if for any path leading to any goal state at least one action from the set of interesting actions has to be fired.

▶ **Definition 8** (Interesting Actions). *Let $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ be a GLTS and $s \in \mathcal{S}$ a state. A set of actions $A_s(Goal) \subseteq A$ is called an* interesting set *of actions for s and $Goal$ if whenever $s \notin Goal$, $w = a_1 \cdots a_n \in A^*$, $s \xrightarrow{w} s'$, and $s' \in Goal$ then there exists $1 \leq i \leq n$ s.t. $a_i \in A_s(Goal)$.*

▶ **Example 9.** In Figure 1 we see an example of a GLTS $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ where $\mathcal{S} = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$ is the states denoted by a circle, $A_1 = \{a, b, c\}$ is the player 1 actions, $A_2 = \{d\}$ is the player 2 actions, and $\rightarrow$ is denoted by the solid and dashed lines between states and labelled with a corresponding action for player 1 and 2, respectively. Let $Goal = \{s_6\}$.

We consider different proposals for a set of interesting actions for the state s_1. The set $\{b\}$ is an interesting set of actions in s_1 since the goal state s_6 cannot be reached without firing b at least once. Furthermore, the sets $\{a\}$ and $\{c\}$ are also sets of interesting actions for the state s_1.

Player 1 has to consider his safe actions. A player 1 action is *safe* in a given player 1 state if by firing it before any sequence of player 1 actions excluding the safe action then it will never reach a player 2 state.

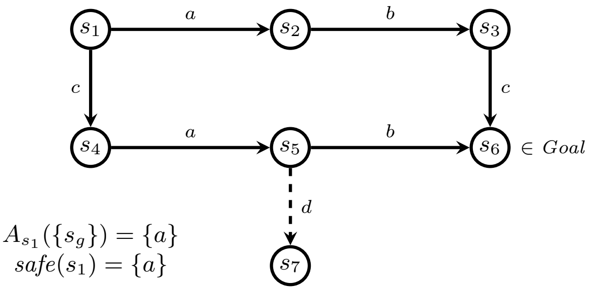


Figure 1 Example of safe and interesting sets of actions for a state s_1.

▶ **Definition 10** (Safe Action). *Let $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ be a GLTS and $s \in \mathcal{S}$ a state s.t. $en_2(s) = \emptyset$. An action $a \in A_1 \cap en_1(s)$ is* safe *in s if whenever $w \in (A_1 \setminus \{a\})^*$ and $s \xrightarrow{w} s'$ and $en_2(s') = \emptyset$ and $s \xrightarrow{aw} s''$ then $en_2(s'') = \emptyset$. The set of all safe actions for s is written as $safe(s)$.*

▶ **Example 11.** Consider again the GLTS in Figure 1. We reasoned in Example 9 that the set $\{b\}$ is an interesting set of actions in the state s_1. However, b is not a safe player 1 action in s_1 since by definition b has to be enabled at s_1 to be safe. The enabled actions at s_1 is $en(s_1) = \{a, c\}$, and between these two actions only a is safe. The action c is not safe since we have $s_1 \xrightarrow{a} s_2$ and $en_2(s_2) = \emptyset$ but $s_1 \xrightarrow{ca} s_5$ and $en_2(s_5) \neq \emptyset$. It is clear from the figure that s_1 is a winning state for player 1 with a as the action player 1 should choose in order to win.

3 Stable Reduction

A reduction St provides at each state a set of actions which are sufficient to fire such that a certain property is preserved in the reduced game. In the game setting, we have to guarantee the preservation of winning strategies for both players in the game. In what follows, we shall introduce a number of conditions that preserve winning strategies and we call such a reduction a *stable* one.

For the remainder of the section let $s \in S$ be a state and $Goal \subseteq \mathcal{S}$ be a set of goal states, and let $A_s(Goal)$ be a fixed set of interesting actions for s and $Goal$.

▶ **Definition 12** (Stable Strategy Conditions). *A reduction St is called* stable *if St satisfies for every $s \in S$ Conditions **I**, **W**, **R**, **G1**, **G2**, **S**, **C**, and **D**.*

I *If $en_1(s) \neq \emptyset$ and $en_2(s) \neq \emptyset$ then $en(s) \subseteq St(s)$.*

W *For all $w \in \overline{St(s)}^*$ and all $a \in St(s)$ if $s \xrightarrow{wa} s'$ then $s \xrightarrow{aw} s'$.*

R *$A_s(Goal) \subseteq St(s)$*

G1 *For all $w \in \overline{St(s)}^*$ if $en_2(s) = \emptyset$ and $s \xrightarrow{w} s'$ then $en_2(s') = \emptyset$.*

G2 *For all $w \in \overline{St(s)}^*$ if $en_1(s) = \emptyset$ and $s \xrightarrow{w} s'$ then $en_1(s') = \emptyset$.*

S *$en_1(s) \cap St(s) \subseteq safe(s)$ or $en_1(s) \subseteq St(s)$*

C *For all $a \in A_2$ if there exists $w \in A_2^\omega$ s.t. $s \xrightarrow{w}$ and a occurs infinitely often in w then $a \in St(s)$.*

D *If $en_2(s) \neq \emptyset$ then there exists $a \in en_2(s) \cap St(s)$ s.t. for all $w \in \overline{St(s)}^*$ where $s \xrightarrow{w} s'$ we have $a \in en_2(s')$.*

If s is a mixed state then Condition **I** ensures all enabled actions are included in the reduction. That is, we do not attempt to reduce the state space from this state. Condition **W** ensures that we can swap the ordering of firing actions, such that firing the actions included in the reduction first still ensures we can reach a given state, i.e. they are independent. Condition **R** ensures that a goal state cannot be reached solely by exploring actions not in the reduction, i.e. reachability of paths to goal states are preserved in the reduction. Conditions **G1** and **G2** ensure that if a state is a player 1 (or player 2) state then a player 2 (or player 1) state cannot be reached solely by exploring actions not in the reduction, i.e. reachability of paths to mixed states and opposing player states are preserved in the reduction. Condition **S** ensures either that all stubborn player 1 actions are also safe and if this is not the case then all player 1 actions are included in the reduction. Condition **C** preserves infinite paths on which only player 2 actions are fired. Condition **D** ensures that at least one player 2 action cannot be disabled solely by exploring actions not in the reduction.

▶ **Example 13.** In Figure 2 we see an example of a GLTS $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ where $\mathcal{S} = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9, s_{10}\}$ are the states denoted by a circle, $A_1 = \{a, b, c, d\}$ is the player 1 actions, $A_2 = \{e\}$ is the player 2 action, and $\rightarrow$ is denoted by the solid and dashed lines between states and labelled with a corresponding action for player 1 and 2, respectively. Let $Goal = \{s_8\}$ be a fixed set of goal states for this GLTS and $A_{s_1}(Goal) = \{a\}$ be a set of interesting actions. Thick lines indicate transitions and states that are preserved by a stable reduction St, while thin lines indicates transitions and states that are removed by the same reduction. For state s_1 we have $St(s_1) = \{a, c\}$ as it is sufficient to satisfy the stable reduction conditions. We satisfy **G1** since c has to be fired once to reach s_7. For $s_1 \xrightarrow{ba} s_5$ and $s_1 \xrightarrow{bc} s_7$ we also have $s_1 \xrightarrow{ab} s_5$ and $s_1 \xrightarrow{cb} s_7$, so **W** is satisfied. Clearly $St(s_1)$ is an interesting set since $A_{s_1}(Goal) \subseteq St(s_1)$, so **R** is satisfied. Condition **S** is satisfied since $St(s_1) \cap en(s_1) \subseteq safe(s_1)$. We have that **I**, **G2**, **C**, and **D** are satisfied as well since their antecedents are not true.

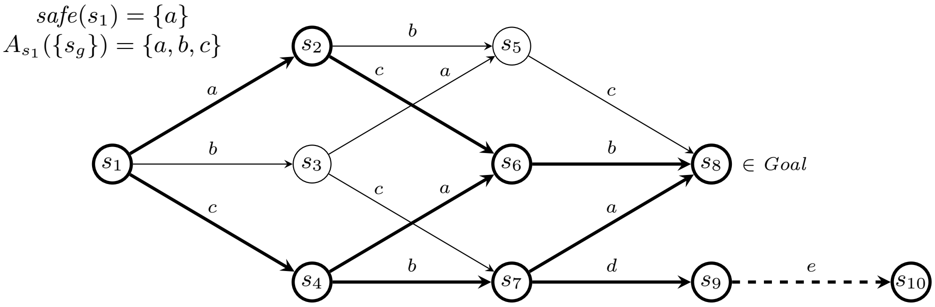


Figure 2 Example of a stable reduction for a state s_1.

We shall first notice the fact that if a goal state is reachable from some state, then the state has at least one enabled action that is also in the stubborn set.

▶ **Lemma 14.** *Let $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ be a GLTS, St a reduction that satisfies Conditions **W** and **R**, and $s \in \mathcal{S}$ a state. If there exists $w \in A^*$ s.t. $s \xrightarrow{w} s'$ and $s' \in Goal$ then $St(s) \cap en(s) \neq \emptyset$.*

The correctness of stable stubborn reductions is proved by the next two lemmas. Both lemmas are proved by induction on the depth of a winning strategy for player 1 in the game.

▶ **Lemma 15.** *Let $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ be a GLTS and St a stable reduction. For all $s \in \mathcal{S}$ if state s is winning for player 1 in G then state s is winning for player 1 in G_{St}.*

▶ **Lemma 16.** *Let $G = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ be a GLTS and St a stable reduction. For all $s \in \mathcal{S}$ if state s is winning for player 1 in G_{St} then state s is winning for player 1 in G.*

We can now present the main theorem showing that stable reductions preserve the winning strategies of both players in the game.

▶ **Theorem 17** (Strategy Preservation for GLTS)**.** *Let $G = (\mathcal{S}, A_1, A_2, \rightarrow)$ be a GLTS and St a stable reduction. For all $s \in \mathcal{S}$ state s is winning for player 1 in G iff state s is winning for player 1 in G_{St}.*

Moreover, for non-mixed games we can simplify the conditions of stable reductions by removing the requirement on safe actions.

▶ **Theorem 18** (Strategy Preservation for Non-Mixed GLTS)**.** *Let $G = (\mathcal{S}, A_1, A_2, \rightarrow)$ be a non-mixed GLTS and St a stable reduction with Condition **S** excluded. For all $s \in \mathcal{S}$ state s is winning for player 1 in G iff state s is winning for player 1 in G_{St}.*

4 Stable Reductions on Petri Net Games

We now introduce the formalism of Petri net games and show how to algorithmically construct stable reductions in a syntax-driven manner.

▶ **Definition 19** (Petri Net Game)**.** *A Petri net game is a tuple $N = (P, T_1, T_2, W, I)$ where P and $T = T_1 \uplus T_2$ are finite sets of places and transitions, respectively, such that $P \cap T = \emptyset$ and where transitions are partitioned into player 1 and player 2 transitions, $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}^0$ is a weight function for regular arcs, and $I : (P \times T) \rightarrow \mathbb{N}^\infty$ is a weight function for inhibitor arcs. A* marking *M on N is a function $M : P \rightarrow \mathbb{N}^0$. The set $\mathcal{M}(N)$ is the set of all markings for N.*

For the rest of this section, let $N = (P, T_1, T_2, W, I)$ be a fixed Petri net game such that $T = T_1 \cup T_2$. Let us first fix some useful notation. For a place or transition x, we denote the *preset* of x as ${}^\bullet x = \{y \in P \cup T \mid W((y, x)) > 0\}$, and the *postset* of x as $x^\bullet = \{y \in P \cup T \mid W((x, y)) > 0\}$. For a transition t, we denote the *inhibitor preset* of t as ${}^\circ t = \{p \in P \mid I((p, t)) \neq \infty\}$, and the *inhibitor postset* of a place p as $p^\circ = \{t \in T \mid I((p, t)) \neq \infty\}$. For a place p we define the *increasing preset* of p, containing all transitions that increase the number of tokens in p, as ${}^+p = \{t \in {}^\bullet p \mid W((t, p)) > W((p, t))\}$, and similarly the *decreasing postset* of p as $p^- = \{t \in p^\bullet \mid W((t, p)) < W((p, t))\}$. For a transition t we define the *decreasing preset* of t, containing all places that have their number of tokens decreased by t, as ${}^-t = \{p \in {}^\bullet t \mid W((p, t)) > W((t, p))\}$, and similarly the *increasing postset* of t as $t^+ = \{p \in t^\bullet \mid W((p, t)) < W((t, p))\}$. For a set X of either places or transitions, we extend the notation as ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x \in X} x^\bullet$, and similarly for the other operators.

A Petri net $N = (P, T_1, T_2, W, I)$ defines a GLTS $G(N) = (\mathcal{S}, A_1, A_2, \rightarrow, Goal)$ where $\mathcal{S} = \mathcal{M}(N)$ is the set of all markings, $A_1 = T_1$ is the set of player 1 actions, $A_2 = T_2$ is the set of player 2 actions, $M \xrightarrow{t} M'$ whenever for all $p \in P$ we have $M(p) \geq W((p, t))$, $M(p) < I((p, t))$ and $M'(p) = M(p) - W((p, t)) + W((t, p))$, and $Goal \in \mathcal{M}(N)$ is the set of goal markings, described by a simple reachability logic formula defined below.

By analysing the increasing presets and postsets, we can identify a sufficient condition for a transition to be safe.

Table 1 Increasing and decreasing transitions for expression $e \in E_N$.

Expression e	$incr_M(e)$	$decr_M(e)$
c	$\emptyset$	$\emptyset$
p	${}^{+}p$	p^{-}
$e_1 + e_2$	$incr_M(e_1) \cup incr_M(e_2)$	$decr_M(e_1) \cup decr_M(e_2)$
$e_1 - e_2$	$incr_M(e_1) \cup decr_M(e_2)$	$decr_M(e_1) \cup incr_M(e_2)$
$e_1 \cdot e_2$	$incr_M(e_1) \cup decr_M(e_1) \cup$ $incr_M(e_2) \cup decr_M(e_2)$	$incr_M(e_1) \cup decr_M(e_1) \cup$ $incr_M(e_2) \cup decr_M(e_2)$

▶ **Lemma 20** (Safe Transition). *Let $N = (P, T_1, T_2, W, I)$ be a Petri net game and $t \in T$ a transition. If $t^+ \cap {}^\bullet T_2 = \emptyset$ and ${}^-t \cap {}^\circ T_2 = \emptyset$ then t is safe in any marking of N.*

Let E_N be the set of marking expressions in N given by the abstract syntax (here e ranges over E_N):

$$e ::= \; c \mid p \mid e_1 \oplus e_2$$

where $c \in \mathbb{N}^0$, $p \in P$, and $\oplus \in \{+, -, *\}$. An expression $e \in E_N$ is evaluated relatively to a marking $M \in \mathcal{M}(N)$ by the function $eval_M : E_N \to \mathbb{Z}$ where $eval_M(c) = c$, $eval_M(p) = M(p)$ and $eval_M(e_1 \oplus e_2) = eval_M(e_1) \oplus eval_M(e_2)$.

In Table 1 we define the functions $incr_M : E_N \to 2^T$ and $decr_M : E_N \to 2^T$ that, given an expression $e \in E_N$, return the set of transitions that can (when fired) increase resp. decrease the evaluation of e.

▶ **Lemma 21** ([4]). *Let $N = (P, T_1, T_2, W, I)$ be a Petri net and $M \in \mathcal{M}(N)$ a marking. Let $e \in E_N$ and let $M \xrightarrow{w} M'$ where $w = t_1 t_2 \ldots t_n \in T^*$.*

- *If $eval_M(e) < eval_{M'}(e)$ then there is i, $1 \leq i \leq n$, such that $t_i \in incr_M(e)$.*
- *If $eval_M(e) > eval_{M'}(e)$ then there is i, $1 \leq i \leq n$, such that $t_i \in decr_M(e)$.*

We can now define the set of reachability formulae Φ_N that evaluate over the markings in N as follows:

$$\varphi ::= \; true \mid false \mid t \mid e_1 \bowtie e_2 \mid deadlock \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \neg\varphi$$

where $e_1, e_2 \in E_N$, $t \in T$ and $\bowtie \in \{<, \leq, =, \neq, >, \geq\}$.

The satisfaction relation for a formula $\varphi \in \Phi_N$ in a marking M is defined as expected:

$$\begin{aligned}
&M \models true && \\
&M \models t && \text{iff } t \in en(M) \\
&M \models e_1 \bowtie e_2 && \text{iff } eval_M(e_1) \bowtie eval_M(e_2) \\
&M \models deadlock && \text{iff } en(M) = \emptyset \\
&M \models \varphi_1 \wedge \varphi_2 && \text{iff } M \models \varphi_1 \text{ and } M \models \varphi_2 \\
&M \models \varphi_1 \vee \varphi_2 && \text{iff } M \models \varphi_1 \text{ or } M \models \varphi_2 \\
&M \models \neg\varphi && \text{iff } M \not\models \varphi
\end{aligned}$$

Our aim is to be able to preserve at least one execution to the set $Goal = \{M \in \mathcal{M}(N) \mid M \models \varphi\}$ for a given formula φ describing the set of goal markings. In order to achieve this, we define the set of interesting transitions $A_M(\varphi)$ for a formulae φ so that any firing sequence of transitions from a marking that does not satisfy φ leading to a marking that satisfies φ must contain at least one interesting transition. Table 2 provides the definition of $A_M(\varphi)$

Table 2 Interesting transitions of φ (assuming $M \not\models \varphi$, otherwise $A_M(\varphi) = \emptyset$).

φ	$A_M(\varphi)$	$A_M(\neg\varphi)$
$deadlock$	$({}^\bullet t)^- \cup {}^+({}^\circ t)$ for some $t \in en(M)$	$\emptyset$
t	${}^+p$ for some $p \in {}^\bullet t$ where $M(p) < W((p,t))$ or p^- for some $p \in {}^\circ t$ where $M(p) \geq I((p,t))$	$({}^\bullet t)^- \cup {}^+({}^\circ t)$
$e_1 < e_2$	$decr_M(e_1) \cup incr_M(e_2)$	$A_M(e_1 \geq e_2)$
$e_1 \leq e_2$	$decr_M(e_1) \cup incr_M(e_2)$	$A_M(e_1 > e_2)$
$e_1 > e_2$	$incr_M(e_1) \cup decr_M(e_2)$	$A_M(e_1 \leq e_2)$
$e_1 \geq e_2$	$incr_M(e_1) \cup decr_M(e_2)$	$A_M(e_1 < e_2)$
$e_1 = e_2$	$decr_M(e_1) \cup incr_M(e_2)$ if $eval_M(e_1) > eval_M(e_2)$ $incr_M(e_1) \cup decr_M(e_2)$ if $eval_M(e_1) < eval_M(e_2)$	$A_M(e_1 \neq e_2)$
$e_1 \neq e_2$	$incr_M(e_1) \cup decr_M(e_1) \cup incr_M(e_2) \cup decr_M(e_2)$	$A_M(e_1 = e_2)$
$\varphi_1 \land \varphi_2$	Defined in Equation (1)	$A_M(\neg\varphi_1 \lor \neg\varphi_2)$
$\varphi_1 \lor \varphi_2$	$A_M(\varphi_1) \cup A_M(\varphi_2)$	$A_M(\neg\varphi_1 \land \neg\varphi_2)$

that is similar to the one presented in [4] for the non-game setting, except for the conjunction where we in our setting use Equation (1) that provides an optimisation for Condition **S** and possibly ends with a smaller set of interesting transitions.

$$A_M(\varphi_1 \land \varphi_2) = \begin{cases} A_M(\varphi_1) & \text{if } M \models \varphi_2 \\ A_M(\varphi_2) & \text{if } M \models \varphi_1 \\ A_M(\varphi_1) & \text{if } M \not\models \varphi_1 \text{ and } A_M(\varphi_1) \subseteq \mathit{safe}(M) \\ A_M(\varphi_2) & \text{if } M \not\models \varphi_2 \text{ and } A_M(\varphi_2) \subseteq \mathit{safe}(M) \\ A_M(\varphi_1) \cup A_M(\varphi_2) & \text{otherwise} \end{cases} \qquad (1)$$

The desired property of the set of interesting transitions is formulated below.

▶ **Lemma 22.** *Let $N = (P, T_1, T_2, W, I)$ be a Petri net, $M \in \mathcal{M}(N)$ a marking, and $\varphi \in \Phi_N$ a formula. If $M \not\models \varphi$ and $M \xrightarrow{w} M'$ where $w \in \overline{A_M(\varphi)}^*$ then $M' \not\models \varphi$.*

We shall now discuss a method for detecting the impossibility of infinite firing sequences consisting of purely player 2 transitions. Let $Fin \subseteq P \cup T_2$ be the smallest set that for every $p \in P$ and every $t \in T_2$ satisfies:

1. $p \in Fin$ whenever $W((p,t)) > W((t,p))$ for every $t \in {}^\bullet p \cap T_2$,
2. $t \in Fin$ whenever ${}^-t \cap Fin \neq \emptyset$, and
3. $p \in Fin$ whenever ${}^\bullet p \cap T_2 \subseteq Fin$.

It is easy to observe that by performing any infinite firing sequence of T_2 transitions, only finitely many tokens can be added to any place from Fin and the infinite firing sequence contains only finitely many occurrences of any transition from Fin. We let $Inf = (P \cup T_2) \setminus Fin$ denote the complement of the set Fin. Note that Fin and Inf only has to be computed once as it is independent of any specific marking.

In Algorithm 1 we present an overapproximation algorithm for detecting cycles of player 2 transitions. The algorithm first checks for orphan transitions (with empty preset) and includes them to the cycle transitions. Then, for a given marking M, it overapproximates the set *MarkedPlaces* of possible places that can be marked by firing T_2 transitions from M. Finally, every transition from the set Inf that has its preset marked is added to the set of possible infinite cycle transitions.

Algorithm 1: *cycle*(N, M): Overapproximation algorithm for computing the set of transitions that may appear in infinite player 2 computations.

```
input  : $N = (P, T_1, T_2, W, I)$ and $M \in \mathcal{M}(N)$
output : If a transition $t \in T_2$ appears infinitely often on some infinite firing
         sequence of $T_2$ transitions from the marking $M$ then $t \in cycle(N, M)$.
$CycleTransitions := \emptyset$;
foreach $t \in T_2$ do
    if ${}^\bullet t = \emptyset$ then
        $CycleTransitions := CycleTransitions \cup \{t\}$;
$FreshPlaces := \{p \in P \mid M(p) > 0\}$; $MarkedPlaces := \emptyset$;
while $MarkedPlaces \neq FreshPlaces$ do
    $MarkedPlaces := FreshPlaces$;
    foreach $p \in P \setminus MarkedPlaces$ do
        if $\exists t \in T_2.\ p \in t^\bullet \wedge {}^\bullet t \subseteq MarkedPlaces$ then
            $FreshPlaces := FreshPlaces \cup \{p\}$;
foreach $t \in Inf \cap T_2$ do
    if ${}^\bullet t \subseteq MarkedPlaces$ then
        $CycleTransitions := CycleTransitions \cup \{t\}$;
return $CycleTransitions$;
```

▶ **Lemma 23.** *Let $N = (P, T_1, T_2, W, I)$ be a Petri net game and $M \in \mathcal{M}(N)$. Algorithm 1 terminates and if there exists $w \in T_2^\omega$ s.t. $M \xrightarrow{w}$ where $t \in T_2$ occurs infinitely often in w then $t \in cycle(N, M)$.*

We can now provide a list of syntactic conditions that guarantee the stability of a given reduction.

▶ **Theorem 24** (Stable Reduction Preserving Closure)**.** *Let $N = (P, T_1, T_2, W, I)$ be a Petri net game, φ a formula, and St a reduction of $G(N)$ such that for all $M \in \mathcal{M}(N)$ the following conditions hold.*

1. *If $en_1(M) \neq \emptyset$ and $en_2(M) \neq \emptyset$ then $en(M) \subseteq St(M)$.*
2. *If $en_1(M) \cap St(M) \not\subseteq safe(M)$ then $en_1(M) \subseteq St(M)$.*
3. *$A_M(\varphi) \subseteq St(M)$*
4. *If $en_1(M) = \emptyset$ then $T_1 \subseteq St(M)$.*
5. *If $en_2(M) = \emptyset$ then $T_2 \subseteq St(M)$.*
6. *$cycle(N, M) \subseteq St(M)$*
7. *For all $t \in St(M)$ if $t \notin en(M)$ then either*
   - **a.** *there exists $p \in {}^\bullet t$ s.t. $M(p) < W((p, t))$ and ${}^+p \subseteq St(s)$, or*
   - **b.** *there exists $p \in {}^\circ t$ s.t. $M(p) \geq I((p, t))$ and $p^- \subseteq St(s)$.*
8. *For all $t \in St(M)$ if $t \in en(M)$ then*
   - **a.** *for all $p \in {}^-t$ we have $p^\bullet \subseteq St(M)$, and*
   - **b.** *for all $p \in t^+$ we have $p^\circ \subseteq St(M)$.*
9. *If $en_2(M) \neq \emptyset$ then there exists $t \in en_2(M) \cup St(M)$ s.t. $({}^\bullet t)^- \cup {}^+({}^\circ t) \subseteq St(M)$.*

*Then St satisfies **I**, **W**, **R**, **G1**, **G2**, **S**, **C**, and **D**.*

Algorithm 2: Computation of $St(M)$ for some stable reduction St.

```
input  : A Petri net game $N = (P, T_1, T_2, W, I)$ and $M \in \mathcal{M}(N)$ and formula $\varphi$
output : $X \subseteq T$ where $X$ is a stable stubborn set for $M$
if $en(M) = \emptyset$ then
    return $T$;
if $en_1(M) \neq \emptyset \wedge en_2(M) \neq \emptyset$ then
    return $T$;
$Y := \emptyset$;
if $en_1(M) = \emptyset$ then
    Pick any $t \in en_2(M)$;
    $Y := T_1 \cup ({}^\bullet t)^- \cup {}^+({}^\circ t)$;
    $Y := Y \cup cycle(N, M)$;
else
    $Y := T_2$;
$Y := Y \cup A_M(\varphi)$;
$X := Saturate(Y)$;
if $X \cap en_1(M) \nsubseteq safe(M)$ then
    return $T$;
return $X$;
```

In Algorithm 2 we provide a pseudocode for calculating stubborn sets for a given marking. The algorithm calls Algorithm 3 that saturates a given set to satisfy Conditions 7 and 8.

▶ **Theorem 25.** *Algorithm 2 terminates and returns $St(M)$ for some stable reduction St.*

▶ Remark 26. In the actual implementation of the algorithm, we first saturate only over the set of interesting transitions and in the case that $Saturate(A_M(\varphi)) \cap en(M) = \emptyset$, we do not explore any of the successors of the marking M as we know that no goal marking can be reached from M (this follows from Lemma 14).

5 Implementation and Experiments

We extend the Petri net verification engine `verifypn` [12], a part of TAPAAL tool suite [6], to experimentally demonstrate the viability of our approach. The synthesis algorithm for solving Petri net games is an adaptation of the dependency graph fixed-point computation from [15, 14] that we reimplement in `C++` while utilising PTries [13] for efficient state storage. The source code is available under GPLv3 [3]. We conduct a series of experiments using the following scalable case studies.

- In *Autonomous Intersection Management* (AIM) vehicles move at different speeds towards an intersection and we want to ensure the absence of collisions. We model the problem as a Petri net game and refer to each instance as AIM-W-X-Y-Z where W is the number of intersections with lanes of length X, Z is the number of cars, and Y is the number of different speeds for each car. The controller assign speeds to cars while the environment aims to cause a collision. The goal marking is where all cars reach their destinations while there are no collisions.
- We reformulate the classical *Producer Consumer System* (PCS) as a Petri net game. In each instance PCS-N-K the total of N consumers (controlled by the environment) and N producers (controlled by the controller) share N buffers. Each consumer and producer has

Algorithm 3: *Saturate*(Y).

```
$X := \emptyset$;
while $Y \neq \emptyset$ do
    Pick any $t \in Y$;
    if $t \notin en(M)$ then
        if $\exists p \in {}^\bullet t.\ M(p) < W((p,t))$ then
            Pick any $p \in {}^\bullet t$ s.t. $M(p) < W((p,t))$;
            $Y := Y \cup ({}^+p \setminus X)$;
        else
            Pick any $p \in {}^\circ t$ s.t. $M(p) \geq I((p,t))$;
            $Y := Y \cup (p^- \setminus X)$;
    else
        $Y := Y \cup ((({}^-t)^\bullet \cup (t^+)^\circ) \setminus X)$;
    $X := X \cup \{t\}$;
    $Y := Y \setminus \{t\}$;
return $X$;
```

a fixed buffer to consume/produce from/to, and each consumer/producer has K different randomly chosen consumption/production rates. The game alternates in rounds where the player choose for each consumer/producer appropriate buffers and rates. The goal of the game is to ensure that the consumers have always enough products in the selected buffers while at the same time the buffers have limited capacity and may not overflow.

- The *Railway Scheduling Problem* contains four instances modelling the Danish train station Lyngby and three of its smaller variants. The scheduling problem, including the station layout, was originally described as a game in [16] and each instance is annotated by a number N representing the number of trains that migrate through the railway network. The controller controls the lights and switches, while the environment moves the trains. The goal of the controller is to make sure that all trains reach (without any collisions) their final destinations.
- The *Nim* (NIM-K-S) Petri net game was described in [22] as a two player game where the players in rounds repeatedly remove between 1 and K pebbles from an initial stack containing S pebbles. The player that has a turn and empty stack of pebbles loses. In our (equivalent) model, we are instead adding pebbles and the player that first adds to or above the given number S loses.
- The *Manufacturing Workflow* (MW) contains instances of a software product line Petri net model presented in [20]. The net describes a series of possible ways of configuring a product (performed by the environment) while the controller aims to construct a requested product. The model instance MW-N contains N possible choices of product features.
- The *Order Workflow* (OW) Petri net game model is taken from [8] and the goal of the game is to synthesise a strategy that guarantees a workflow soundness, irrelevant of the choices made by the environment. We scale the workflow by repeatedly re-initialising the workflow N times (denoted by OW-N).

All experimental evaluation is run on AMD Opteron 6376 Processors with 120 GB memory limitation and 12 hours timeout (we measure only the execution time without the parsing time of the models). We use for all experiments the depth first search strategy and we only

Table 3 Experiments with and without partial order reduction (POR and NORMAL).

	Time (seconds)		Markings ×1000		Reduction	
Model	NORMAL	POR	NORMAL	POR	%Time	%Markings
AIM-13-100-6-11	117.9	46.6	1702	514	60	70
AIM-13-100-6-16	173.9	63.2	2464	746	64	70
AIM-13-150-9-16	337.0	219.9	3696	2454	35	34
AIM-13-150-9-21	408.0	294.1	4853	3331	28	31
AIM-14-150-9-16	449.9	278.4	4259	2865	38	33
AIM-15-150-9-16	534.5	384.9	4861	3204	28	34
PCS-2-2	24.0	19.9	9629	6554	17	32
PCS-2-3	116.1	90.9	61990	39114	22	37
PCS-2-4	399.1	283.3	240510	145109	29	40
LyngbySmall-2	3.4	1.0	1803	444	71	75
LyngbySmall-3	23.1	26.1	11473	10701	-13	7
LyngbySmall-4	144.3	193.3	65371	70008	-34	-7
Lyngby-2	3292.0	215.5	1511749	87214	93	94
NIM-5-49500	9.2	3.4	5054	892	63	82
NIM-7-49500	32.7	3.9	24039	1159	88	95
NIM-9-49500	165.1	4.7	114235	1522	97	99
NIM-11-49500	710.7	8.2	533516	1877	99	100
MW-40	735.3	0.2	69439	9	100	100
MW-50	1952.0	0.2	135697	11	100	100
MW-60	4417.0	0.3	234570	13	100	100
OW-10000	0.9	0.7	320	240	22	25
OW-100000	11.1	7.8	3200	2400	30	25
OW-1000000	137.7	109.8	32000	24000	20	25

report the examples where the algorithms both with and without partial order reduction returned a result within the time and memory limits. We provide a reproducibility package with all models and experimental data [3].

5.1 Results

Table 3 shows the experimental evaluation, displaying the relative gain in computation time (in seconds) without and with partial order reduction as well in the number of explored markings (in thousands). The results demonstrate significant reductions across all models, in some cases like in NIM and MW even of several degrees of magnitude. Other models like PCS, AIM and OW show a moderate but significant reduction. We observe that the time reduction is generally only few percent smaller than the reduction in the number of explored markings, showing typically between 2% to 20% overhead for computing (on-the-fly) the stubborn sets. For two instances of the LyngbySmall models we notice an actual slowdown in the running time where for LyngbySmall-3 the number of reduced markings is less significant and it results in 13% slowdown, partially due to the overhead for computing stubborn sets but also because the search strategy changed and this resulted in the fact that we have to search a larger portion of the generated dependency graph before we obtain a conclusive answer. This effect is, in particular, observed in the LyngbySmall-4 example where the number of markings stored when applying partial order reduction actually exceeds those where no reduction is applied. Nevertheless, in general the experiments confirm the practical applicability of partial order reduction for 2-player games with only minimal overhead for computing the stubborn sets.

6 Conclusion

We generalised the partial order reduction technique based on stubborn sets from plain reachability to a game theoretical setting. This required a nontrivial extension of the classical conditions on stubborn sets so that a state space reduction can be achieved for both players in the game. In particular, the computation of the stubborn sets for player 2 (uncontrollable transitions) needed a new insight on how to handle and efficiently approximate the existence of infinite runs with player 2 transitions only. We proved the correctness of our approach and instantiated it to the case of Petri net games. We provided (to the best of our knowledge) the first implementation of partial order reduction for Petri net games and made it available as a part of the model checker TAPAAL. The experiments show promising results on a number of case studies, achieving in general a substantial state space reduction with only a small overhead for computing the stubborn sets. In the future work, we plan to combine our contribution with a recent insight on how to effectively use partial order reduction in the timed setting [4] in order to extend our framework to general timed games.

References

1 N. Alechina, N. Bulling, S. Demri, and B. Logan. On the complexity of resource-bounded logics. In *Reachability Problems*, volume 9899 of *LNCS*, pages 36–50. Springer-Verlag, 2016.

2 B. Bérard, S. Haddad, M. Sassolas, and N. Sznajder. Concurrent Games on VASS with Inhibition. In *International Conference on Concurrency Theory*, volume 7454 of *LNCS*, pages 39–52. Springer-Verlag, 2012.

3 F. M. Bønneland, P. G. Jensen, K. G. Larsen, M. Mũniz, and J. Srba. Artifact for "Partial Order Reduction for Reachability Games", June 2019. doi:10.5281/zenodo.3252104.

4 F.M. Bønneland, J. Dyhr, P.G. Jensen, M. Johannsen, and J. Srba. Start Pruning When Time Gets Urgent: Partial Order Reduction for Timed Systems. In *Computer Aided Verification*, volume 10981 of *LNCS*, pages 527–546. Springer-Verlag, 2018.

5 A.E. Dalsgaard, S. Enevoldsen, P. Fogh, L.S. Jensen, P.G. Jensen, T.S. Jepsen, I. Kaufmann, K.G. Larsen, S.M. Nielsen, M.Chr. Olesen, S. Pastva, and J. Srba. A Distributed Fixed-Point Algorithm for Extended Dependency Graphs. *Fundamenta Informaticae*, 161(4):351–381, 2018. doi:10.3233/FI-2018-1707.

6 A. David, L. Jacobsen, M. Jacobsen, K.Y. Jørgensen, M.H. Møller, and J. Srba. TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 7214 of *LNCS*, pages 492–497. Springer-Verlag, 2012.

7 R. Davidrajuh. Detecting Existence of Cycles in Petri Nets. In *International Joint Conference SOCO'16-CISIS'16-ICEUTE'16*, volume 527 of *AISC*, pages 376–385. Springer-Verlag, 2016.

8 J. Dehnert and A. Zimmermann. Making Workflow Models Sound Using Petri Net Controller Synthesis. In *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*, volume 3290 of *LNCS*, pages 139–154. Springer-Verlag, 2004.

9 M. Huhn, P. Niebert, and H. Wehrheim. Partial Order Reductions for Bisimulation Checking. In *Foundations of Software Technology and Theoretical Computer Science*, volume 1530 of *LNCS*, pages 271–282. Springer-Verlag, 1998.

10 W. Jamroga, W. Penczek, P. Dembiński, and A. Mazurkiewicz. Towards Partial Order Reductions for Strategic Ability. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS 2018, pages 156–165, Richland, SC, 2018. ACM.

11 E. Javier. *Decidability and Complexity of Petri Net Problems – An Introduction*, volume 1491 of *LNCS*, pages 374–428. Springer-Verlag, 1998.

12 J.F Jensen, T. Nielsen, L.K. Oestergaard, and J. Srba. TAPAAL and Reachability Analysis of P/T Nets. In *Transactions on Petri Nets and Other Models of Concurrency XI*, volume 9930 of *LNCS*, pages 307–318. Springer-Verlag, 2016.

13 P. G. Jensen, K. G. Larsen, and J. Srba. PTrie: Data Structure for Compressing and Storing Sets via Prefix Sharing. In *Proceedings of the 14th International Colloquium on Theoretical Aspects of Computing (ICTAC'17)*, volume 10580 of *LNCS*, pages 248–265. Springer-Verlag, 2017. `doi:10.1007/978-3-319-67729-3_15`.

14 P.G. Jensen, K.G. Larsen, and J. Srba. Real-Time Strategy Synthesis for Timed-Arc Petri Net Games via Discretization. In *International Symposium on Model Checking Software*, volume 9641 of *LNCS*, pages 129–146. Springer-Verlag, 2016.

15 P.G. Jensen, K.G. Larsen, and J. Srba. Discrete and continuous strategies for timed-arc Petri net games. *International Journal on Software Tools for Technology Transfer*, 20(5):529–546, October 2018.

16 P. Kasting, M.R. Hansen, and S. Vester. Synthesis of Railway-Signaling Plans using Reachability Games. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages*, IFL 2016, pages 9:1–9:13, New York, NY, USA, 2016. ACM. `doi:10.1145/3064899.3064908`.

17 A. Laarman and A. Wijs. Partial-Order Reduction for Multi-core LTL Model Checking. In *Hardware and Software: Verification and Testing*, volume 8855 of *LNCS*, pages 267–283. Springer-Verlag, 2014.

18 A. Lehmann, N. Lohmann, and K. Wolf. Stubborn Sets for Simple Linear Time Properties. In *Application and Theory of Petri Nets*, volume 7347 of *LNCS*, pages 228–247. Springer-Verlag, 2012.

19 D. Peled. All from One, One for All: on Model Checking Using Representatives. In *Computer Aided Verification*, volume 697 of *LNCS*, pages 409–423. Springer-Verlag, 1993.

20 F.G. Quintanilla, S. Kubler, O. Cardin, and P. Castagna. Product Specification in a Service-Oriented Holonic Manufacturing System using Petri-Nets. *IFAC Proceedings Volumes*, 46(7):342–347, May 2013. `doi:10.3182/20130522-3-BR-4036.00094`.

21 Y.S. Ramakrishna and S.A. Smolka. Partial-Order Reduction in the Weak Modal Mu-Calculus. In Antoni Mazurkiewicz and Józef Winkowski, editors, *International Conference on Concurrency Theory*, volume 1243 of *LNCS*, pages 5–24. Springer-Verlag, 1997.

22 R. Tagiew. Multi-Agent Petri-Games. In *International Conference on Computational Intelligence for Modeling Control Automation*, volume 10981 of *LNCS*, pages 130–135. IEEE Computer Society, 2008. `doi:10.1109/CIMCA.2008.15`.

23 A. Valmari. A Stubborn Attack on State Explosion. *Formal Methods in System Design*, 1(4):297–322, December 1992.

24 A. Valmari. On-The-Fly Verification with Stubborn Sets. In *Computer Aided Verification*, volume 697 of *LNCS*, pages 397–408. Springer-Verlag, 1993.

25 A. Valmari and H. Hansen. Stubborn Set Intuition Explained. In *Transactions on Petri Nets and Other Models of Concurrency XII*, volume 10470 of *LNCS*, pages 140–165. Springer-Verlag, 2017.

26 B. Willems and P. Wolper. Partial-Order Methods for Model Checking: From Linear Time to Branching Time. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pages 294–303, July 1996. `doi:10.1109/LICS.1996.561357`.

Synthesis of Data Word Transducers

Léo Exibard
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France
Université libre de Bruxelles, Brussels, Belgium

Emmanuel Filiot
Université libre de Bruxelles, Brussels, Belgium

Pierre-Alain Reynier
Aix Marseille Univ, Université de Toulon, CNRS, LIS, Marseille, France

Abstract

In reactive synthesis, the goal is to automatically generate an implementation from a specification of the reactive and non-terminating input/output behaviours of a system. Specifications are usually modelled as logical formulae or automata over infinite sequences of signals (ω-words), while implementations are represented as transducers. In the classical setting, the set of signals is assumed to be finite. In this paper, we consider data ω-words instead, i.e., words over an infinite alphabet. In this context, we study specifications and implementations respectively given as automata and transducers extended with a finite set of registers. We consider different instances, depending on whether the specification is nondeterministic, universal or deterministic, and depending on whether the number of registers of the implementation is given or not.

In the unbounded setting, we show undecidability for both universal and non-deterministic specifications, while decidability is recovered in the deterministic case. In the bounded setting, undecidability still holds for non-deterministic specifications, but can be recovered by disallowing tests over input data. The generic technique we use to show the latter result allows us to reprove some known result, namely decidability of bounded synthesis for universal specifications.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Automata over infinite objects; Theory of computation → Transducers

Keywords and phrases Register Automata, Synthesis, Data words, Transducers

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.24

Related Version An extended version can be found at `https://arxiv.org/abs/1905.03538`.

Funding *Léo Exibard*: Funded by a FRIA fellowship from the F.R.S.-FNRS.
Emmanuel Filiot: Research associate of F.R.S.-FNRS. He is supported by the ARC Project Transform Fédération Wallonie-Bruxelles and the FNRS CDR J013116F and MIS F451019F projects.
Pierre-Alain Reynier: Partly funded by the DeLTA project (ANR-16-CE40-0007).

Acknowledgements We would like to thank Ayrat Khalimov for his remarks and suggestions, which helped improve the quality of the paper.

1 Introduction

Reactive synthesis is an active research domain whose goal is to design algorithmic methods able to construct a reactive system from a specification of its admissible behaviours. Such systems are notoriously difficult to design correctly, and the main appealing idea of synthesis is to automatically generate systems *correct by construction.* Reactive systems are non-terminating systems that continuously interact with the environment in which they are executed, through input and output *signals.* At each time step, the system receives an input signal from a set In and produces an output signal from a set Out. An execution is then modelled as an infinite sequence alternating between input and output signals, i.e., an ω-word in $(\mathsf{In}.\mathsf{Out})^\omega$. Classically, the sets In and Out are *assumed to be finite* and reactive

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 24; pp. 24:1–24:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

systems are modelled as (sequential) *transducers*. Transducers are simple finite-state machines with transitions of type $\mathsf{States} \times \mathsf{In} \to \mathsf{States} \times \mathsf{Out}$, which, at any state, can process any input signal and deterministically produce some output signal, while possibly moving, again deterministically, to a new state. A *specification* is then a language $S \subseteq (\mathsf{In}.\mathsf{Out})^\omega$ telling which are the acceptable behaviours of the system. It is also classically represented as an automaton, or as a logical formula then converted into an automaton. Some regular specifications may not be realisable by any transducer, and the *realisability problem* asks, given a regular specification S, whether there exists a transducer T whose behaviours satisfy S (are included in S). The synthesis problem asks to construct T if it exists.

A typical example of reactive system is that of a server granting requests from a *finite* set of clients C. Requests are represented as the set of input signals $\mathsf{In} = \{(r,i) \mid i \in C\} \cup \{\mathsf{idle}\}$ (client i requests the ressource) and grants by the set of output signals $\mathsf{Out} = \{(g,i) \mid i \in C\} \cup \{\mathsf{idle}\}$ (server grants client i's request). A typical constraint to be imposed on such a system is that every request is eventually granted, which can be represented by the LTL formula $\bigwedge_{i\in C} G((r,i) \to F(g,i))$. The latter specification is realisable for instance by the transducer which outputs (g,i) whenever it reads (r,i) and idle whenever it reads idle.

It is well-known that the realisability problem is decidable for ω-regular specifications, ExpTime-c when represented by parity automata [9] and 2ExpTime-c for LTL specifications [16]. Such positive results have triggered a recent and very active research interest in efficient symbolic methods and tools for reactive synthesis (see e.g. [1]). Extensions of this classical setting have been proposed to capture more realistic scenarios [1]. However, only a few works have considered infinite sets of input and output signals. In the previous example, the number of clients is assumed to be finite, and small. To the best of our knowledge, existing synthesis tools do not handle large alphabets, and it is more realistic to consider an unbounded (infinite) set of client identifiers, e.g. $C = \mathbb{N}$. The goal of this paper is to investigate how reactive synthesis can be extended to handle infinite sets of signals.

Data words are infinite sequences $x_1x_2\dots$ of labelled data, i.e., pairs (σ,d) with σ a label from a finite alphabet, and $d \in \mathbb{N}$. They can naturally model executions of reactive systems over an infinite set of signals. Among other models, *register automata* are one of the main extensions of automata recognising languages of data words [10, 18]. They can use a finite set of registers in which to store data that are read, and to compare the current data with the content of some of the registers (in this paper, we allow comparison of equality). Likewise, transducers can be extended to *register transducers* as a model of reactive systems over data words: a register transducer is equipped with a set of registers, and when reading an input labelled data (σ,d), it can test d for equality with the content of some of its registers, and depending on the result of this test, deterministically assign some of its registers to d and output a finite label β together with the content of one of its registers. Its executions are then data words alternating between input and output labelled data, so register automata can be used to represent specifications, as languages of such data words.

Contributions. We consider two classical semantics for register automata, non-deterministic and universal, both with a parity acceptance condition, which give two classes of register automata respectively denoted NRA and URA. Since NRA are not closed under complement (already over finite data words), NRA and URA define incomparable classes of specifications. The request-grant specification given previously with an infinite number of clients is expressible by an URA [12]: whenever a request is made by client i (labelled data (r,i)), universally trigger a run which stores i in some register and verifies that the labelled data (g,i) eventually occurs in the data word; but no NRA can define it. On the contrary, consider the specification S_0:

"all input data but one are copied on the output, the missing one being replaced by some data which occurred before it", modelled as the set of data sequences $d_1d_1d_2d_2 \dots d_id_jd_{i+1}d_{i+1} \dots$ for all $i \geq 0$ and $j < i$ (finite labels are irrelevant and not represented). S_0 is not definable by any URA (it would require to guess j, which can be arbitrarily smaller than i), but it is expressible by some NRA making this guess.

However, we show (unsurprisingly) that the realisability problem by register transducers of specifications defined by NRA is undecidable. The same negative result also holds for URA, solving an open question raised in [12]. On the positive side, we show that decidability is recovered for deterministic (parity) register automata (DRA). One of the difficulties of register transducer synthesis is that the number of registers needed to realise the specification is, a priori, unbounded with regards to the number of registers of the specification. We show it is in fact not the case for DRA: any specification expressed as a DRA with r registers is realisable by a register transducer iff it is realisable by a transducer with r registers.

A way to obtain decidability is to fix a bound k and to target register transducers with at most k registers. This setting is called *bounded synthesis* in [12], which establishes that bounded synthesis is decidable in 2ExpTime for URA. We show that unfortunately, bounded synthesis is still undecidable for NRA specifications. To recover decidability for NRA, we disallow equality tests on the input data and add a syntactic requirement which entails that on any accepted word, each output data is the content of some register which has been assigned an input data occurring before. This defines a subclass of NRA that we call (input) *test-free* NRA ($\mathsf{NRA_{tf}}$). $\mathsf{NRA_{tf}}$ can express how output data can be obtained from input data (by copying, moving or duplicating them), although they do not have the whole power of register automata on the input nor the output side. Note that the specification S_0 given before is $\mathsf{NRA_{tf}}$-definable. To show that bounded synthesis is decidable for $\mathsf{NRA_{tf}}$, we establish a generic transfer property characterising realisable data word specifications in terms of realisability of corresponding specifications over a finite alphabet, thus reducing to the well-known synthesis problem over a finite alphabet. Such property also allows us to reprove the result of [12], with a rather short proof based on standard results from the theory of register automata, indicating that it might allow to establish decidability for other classes of data specifications. Our results are summarised in Table 1.

Table 1 Decidability status of the problems studied.

	DRA	NRA	URA	$\mathsf{NRA_{tf}}$
Bounded Synthesis	ExpTime (Thm. 13)	Undecidable ($k \geq 1$) (Thm. 3)	2ExpTime ([12] and Thm. 12)	2ExpTime (Thm. 16)
General Case	ExpTime (Thm. 6)	Undecidable (Thm. 2)	Undecidable (Thm. 4)	Open

Related Work. As already mentioned, bounded synthesis of register transducers is considered in [12] where it is shown to be decidable for URA. We reprove this result in a shorter way. Our proof bears some similarities with that of [12], but it seems that our formulation benefits more from the use of existing results. The technique is also more generic and we instantiate it to $\mathsf{NRA_{tf}}$. $\mathsf{NRA_{tf}}$ correspond to the one-way, non-deterministic version of the expressive transducer model of [5], which however does not consider the synthesis problem.

The synthesis problem over infinite alphabets is also considered in [6], in which data represent identifiers and specifications (given as particular automata close to register automata) can depend on equality between identifiers. However, the class of implementations is very expressive: it allows for unbounded memory through a queue data structure. The synthesis problem is shown to be undecidable and a sound but incomplete algorithm is given.

Finally, classical reactive synthesis has strong connections with game theory on finite graphs. Some extension of games to infinite graphs whose vertices are valuations of variables in an infinite data domain have been considered in [8]. Such games are shown to be undecidable and a decidable restriction is proposed, which however does not seem to match our context.

2 Data Words and Register Automata

For a (possibly infinite) set S, we denote by S^ω the set of infinite words over this alphabet. For $1 \leq i \leq j$, we let $u[i{:}j] = u_i u_{i+1} \dots u_j$ and $u[i] = u[i{:}i]$ the ith letter of u. For $u, v \in S^\omega$, we define their *interleaving* $\langle u, v \rangle = u[1]v[1]u[2]v[2]\dots$

Data Words. Let Σ be a finite alphabet and $\mathcal{D}$ a countably infinite set, denoting, all over this paper, a set of elements called *data*. We also distinguish an (arbitrary) data value $\mathsf{d}_0 \in \mathcal{D}$. Given a set R, let τ_0^R be the constant function defined by $\tau_0^R(r) = \mathsf{d}_0$ for all $r \in R$. A *labelled data* (or l-data for short) is a pair $x = (\sigma, d) \in \Sigma \times \mathcal{D}$, where σ is the *label* and d the *data*. We define the projections $\mathsf{lab}(x) = \sigma$ and $\mathsf{dt}(x) = d$. A *data word* over Σ and $\mathcal{D}$ is an infinite sequence of labelled data, i.e. a word $w \in (\Sigma \times \mathcal{D})^\omega$. We extend the projections lab and dt to data words naturally, i.e. $\mathsf{lab}(w) \in \Sigma^\omega$ and $\mathsf{dt}(w) \in \mathcal{D}^\omega$. We denote the set of data words over Σ and $\mathcal{D}$ by $\mathsf{DW}(\Sigma, \mathcal{D})$ (DW when clear from the context). A *data word language* is a subset $L \subseteq \mathsf{DW}(\Sigma, \mathcal{D})$. Note that in this paper, data words are infinite, otherwise they are called *finite data words*, and we denote by $\mathsf{DW}_f(\Sigma, \mathcal{D})$ the set of finite data words.

Register Automata. Register automata are automata recognising data word languages. They were first introduced in [10] as finite-memory automata. Here, we define them in a spirit close to [18], but over infinite words [1], with a parity acceptance condition.

A *register automaton* (RA) is a tuple $\mathcal{A} = (\Sigma, \mathcal{D}, Q, q_0, \delta, R, c)$, where:

- Σ is a finite alphabet of *labels*, $\mathcal{D}$ is an infinite alphabet of *data*
- Q is a finite set of states and $q_0 \in Q$ is the initial state
- R is a finite set of *registers*. We denote $\mathsf{Tst}_R = 2^R$ and $\mathsf{Asgn}_R = 2^R$. [2]
- $c : Q \to \{1, \dots, d\}$, where $d \in \mathbb{N}$ is the number of *priorities*, is the *colouring function*, used to define the acceptance condition
- $\delta \subseteq Q \times \Sigma \times \mathsf{Tst}_R \times \mathsf{Asgn}_R \times Q$ is a set of transitions.

A transition $(q, \sigma, \mathsf{tst}, \mathsf{asgn}, q')$ is also written $q \xrightarrow[\mathcal{A}]{\sigma,\mathsf{tst},\mathsf{asgn}} q'$. We may omit $\mathcal{A}$ in the latter notation. Intuitively such transition means that on input (σ, d) in state q the automaton: it first checks that tst is *exactly* the set of registers containing d: for all $r \in \mathsf{tst}$, d is the current content of register r and for all $r \notin \mathsf{tst}$, d is not in register r, then it assigns d to all the registers in asgn (asgn might be empty), and finally transitions to state q'.

$\mathcal{A}$ is said to be *deterministic* (resp. *complete*) when, given any state, any label and any possible test, at most (resp. at least) one transition can be taken: $\forall q \in Q, \forall \sigma \in \Sigma, \forall \mathsf{tst} \in \mathsf{Tst}_R, \exists^{\leq 1} \mathsf{asgn} \in \mathsf{Asgn}, \exists^{\leq 1} q' \in Q$ such that $q \xrightarrow{\sigma,\mathsf{tst},\mathsf{asgn}} q'$ (resp. $\forall q, \forall \sigma, \forall \mathsf{tst}, \exists^{\geq 1} \mathsf{asgn}, \exists^{\geq 1} q'$ s.t. $q \xrightarrow{\sigma,\mathsf{tst},\mathsf{asgn}} q'$). Since tests are mutually exclusive, this syntactically ensures that for any state and in any register configuration, the transition to take is determined by the input l-data (respectively that a transition can always be taken, regardless the input l-data). The class of deterministic register automata will be denoted DRA.

[1] In the terminology of [13], it corresponds to multiple-assignment register automata with registers initially filled, i.e. the class MF, with the additional requirement that all are filled with the same data.

[2] Those sets are identical but have distinct semantics.

Configurations and Runs. A configuration is a pair $(q,\tau) \in Q \times (R \to \mathcal{D})$. Given $\mathsf{tst} \in \mathsf{Tst}_R$ and $d \in \mathcal{D}$, we say that τ, d satisfies tst, denoted $\tau, d \models \mathsf{tst}$ if $\tau^{-1}(d) = \mathsf{tst}$. Given a transition $t = p \xrightarrow{\sigma,\mathsf{tst},\mathsf{asgn}} p'$, we say that (q,τ) enables t on reading (σ', d) if $q = p$, $\sigma' = \sigma$ and $\tau, d \models \mathsf{tst}$. Let $\mathrm{next}(\tau, \mathsf{asgn}, d)$ be the configuration τ' defined by $\tau'(i) = d$ if $i \in \mathsf{asgn}$, and $\tau'(i) = \tau(i)$ otherwise. We extend this notation to configurations as follows: if $\gamma = (q,\tau)$ enables t on input (σ, d), the *successor* configuration of (q,τ) by t on input (σ, d) is $\mathrm{next}(\gamma, \mathsf{asgn}, d) = (p', \mathrm{next}(\tau, \mathsf{asgn}, d))$. We also write $\mathrm{next}(\gamma, t, \sigma, d)$ to denote the successor of (q,τ) by transition t when (q,τ) enables t on input (σ, d). The *initial configuration* is (q_0, τ_0^R). Then, a *run* over a data word $(\sigma_1, d_1)(\sigma_2, d_2)\dots$ is an infinite sequence of transitions $t_0 t_1 \dots$ such that there exists a sequence of configurations $\gamma_0\gamma_1 \dots = (q_0,\tau_0)(q_1,\tau_1)\dots$ such that γ_0 is initial and for all $i \geq 0$, $\gamma_{i+1} = \mathrm{next}(\gamma_i, t_i, \sigma_i, d_i)$. To a run ρ, we associate its sequence of states $\mathsf{states}(\rho) = q_0 q_1 \dots$

Languages Defined by RA. Given a run ρ, we denote, by a slight abuse of notation, $c(\rho) = \max\{j \mid c(q_l) = j \text{ for infinitely many } q_l \in \mathsf{states}(\rho)\}$ the maximum color that occurs infinitely often in ρ. Then, in the parity acceptance condition, ρ is accepting whenever $c(\rho)$ is even. We consider two dual semantics for RA: non-deterministic (N) and universal (U). Given an RA A, depending on whether it is considered nondeterministic or universal, it recognises $L_N(A) = \{w \mid \text{there exists an accepting run } \rho \text{ on } w\}$ or $L_U(A) = \{w \mid \text{all runs } \rho \text{ on } w \text{ are accepting}\}$.

We denote by NRA (resp. URA) the class of register automata interpreted with a non-deterministic (resp. universal) parity acceptance condition, and given $A \in \mathsf{NRA}$ (resp. $A \in \mathsf{URA}$), we write $L(A)$ instead of $L_N(A)$ (resp. $L_U(A)$). We also denote by DRA the class of deterministic parity register automata.

3 Synthesis of Register Transducers

Specifications, Implementations and the Realisability Problem. Let $\Sigma_{\mathbb{i}}$ and $\Sigma_{\mathbb{o}}$ be two finite alphabets of labels, and $\mathcal{D}$ a countable set of data. A *relational data word* is an element of $w \in [(\Sigma_{\mathbb{i}} \times \mathcal{D}).(\Sigma_{\mathbb{o}} \times \mathcal{D})]^\omega$. Such a word is called relational as it defines a pair of data words in $\mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D}) \times \mathsf{DW}(\Sigma_{\mathbb{o}}, \mathcal{D})$ through the following projections. If $w = x_{\mathbb{i}}^1 x_{\mathbb{o}}^1 x_{\mathbb{i}}^2 x_{\mathbb{o}}^2 \dots$, we let $\mathsf{inp}(w) = x_{\mathbb{i}}^1 x_{\mathbb{i}}^2 \dots$ and $\mathsf{out}(w) = x_{\mathbb{o}}^1 x_{\mathbb{o}}^2 \dots$ We denote by $\mathsf{RW}(\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, \mathcal{D})$ (just RW when clear from the context) the set of relational data words. A *specification* is simply a language $S \subseteq \mathsf{RW}(\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, \mathcal{D})$. An *implementation* is a total function $I : (\Sigma_{\mathbb{i}} \times \mathcal{D})^* \to \Sigma_{\mathbb{o}} \times \mathcal{D}$. We associate to I another function $f_I : \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D}) \to \mathsf{DW}(\Sigma_{\mathbb{o}}, \mathcal{D})$ which, to an input data word $w_{\mathbb{i}} = x_{\mathbb{i}}^1 x_{\mathbb{i}}^2 \dots \in \Sigma_{\mathbb{i}} \times \mathcal{D}$, associates the output data word $f_I(w_{\mathbb{i}}) = x_{\mathbb{o}}^1 x_{\mathbb{o}}^2 \dots$ such that $\forall i \geq 1$, $x_{\mathbb{o}}^i = I(x_{\mathbb{i}}^1 \dots x_{\mathbb{i}}^{i-1})$. I also defines a language of relational data words $L(I) = \{\langle w_{\mathbb{i}}, f_I(w_{\mathbb{i}})\rangle \mid w_{\mathbb{i}} \in \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})\}$.

We say that I *realises* S when $L(I) \subseteq S$, and that S is *realisable* if there exists an implementation realising it. The *realisability problem* consists, given a (finite representation of a) specification S, in checking whether S is realisable. In general, we parameterise this problem by classes of specifications $\mathcal{S}$ and of implementations $\mathcal{I}$, defining the $(\mathcal{S}, \mathcal{I})$-realisability problem, denoted $\textsc{Real}(\mathcal{S}, \mathcal{I})$. Given a specification $S \in \mathcal{S}$, it asks whether S is realisable by some implementation $I \in \mathcal{I}$. We now introduce the classes $\mathcal{S}$ and $\mathcal{I}$ we consider.

Specification Register Automata. In this paper, we consider specifications defined from register automata alternately reading input and output l-data. We assume that the set of states is partitioned into $Q_{\mathbb{i}}$ (called input states, reading only labels in $\Sigma_{\mathbb{i}}$) and $Q_{\mathbb{o}}$ (called output states, reading only labels in $\Sigma_{\mathbb{o}}$), where $q_0 \in Q_{\mathbb{i}}$ and $F \subseteq Q_{\mathbb{i}}$, and such that the transition relation δ alternates between these two sets, i.e. $\delta \subseteq \cup_{\alpha=\mathbb{i},\mathbb{o}}(Q_\alpha \times \Sigma_\alpha \times \mathsf{Tst}_R \times \mathsf{Asgn}_R \times Q_{\overline{\alpha}})$, where $\overline{\mathbb{i}} = \mathbb{o}$ (resp. $\overline{\mathbb{o}} = \mathbb{i}$). We denote by DRA (resp. NRA, URA) the class of specifications defined by deterministic (resp. non-deterministic, universal) parity register automata.

Register Transducers As Implementations. We consider implementations represented as transducers processing data words. A *register transducer* is a tuple $T = (\Sigma_\mathbb{i}, \Sigma_\mathbb{o}, Q, q_0, \delta, R)$ where Q is a finite set of states with initial state q_0, R is a finite set of registers, and $\delta : Q \times \Sigma_\mathbb{i} \times \mathsf{Tst}_R \to \mathsf{Asgn}_R \times \Sigma_\mathbb{o} \times R \times Q$ is the (total) transition function (as before, $\mathsf{Tst}_R = \mathsf{Asgn}_R = 2^R$). When processing an I-data $(\sigma_\mathbb{i}, d)$, T compares d with the content of some of its registers, and depending on the result, moves to another state, stores d in some registers, and outputs some label in $\Sigma_\mathbb{o}$ along with the content of some register $r \in R$.

Let us formally define the semantics of a register transducer T, as an implementation I_T. First, for a finite input data word $w = (\sigma_\mathbb{i}^1, d_\mathbb{i}^1) \dots (\sigma_\mathbb{i}^n, d_\mathbb{i}^n)$ in $(\Sigma_\mathbb{i} \times \mathcal{D})^*$, we denote by (q_i, τ_i) the ith configuration reached by T on w, where (q_0, τ_0) is initial and for all $0 < i < n$, (q_i, τ_i) is the unique configuration such that there exists a transition $\delta(q_{i-1}, \sigma_\mathbb{i}^i, \mathsf{tst}) = (\mathsf{asgn}, \sigma_\mathbb{o}, r, q_i)$ such that $\tau_{i-1}, d_\mathbb{i}^i \models \mathsf{tst}$ and $\tau_i = \mathrm{next}(\tau_{i-1}, d_\mathbb{i}^i, \mathsf{asgn})$. We let $(\sigma_\mathbb{o}^i, d_\mathbb{o}^i) = (\sigma_\mathbb{o}, \tau_i(r))$ and $I_T(w) = (\sigma_\mathbb{o}^n, d_\mathbb{o}^n)$. Then, we denote $f_T = f_{I_T}$ and $L(T) = L(I_T)$. Note that if T is interpreted as a DRA with exactly one transition per output state and whose states are all accepting (i.e. have even maximal parity 2), then $L(I_T)$ is indeed the language of such register automaton. We denote by $\mathsf{RT}[k]$ the class of implementations defined by register transducers with at most k registers, and by $\mathsf{RT} = \bigcup_{k \geq 0} \mathsf{RT}[k]$ the class of implementations defined by register transducers.

Synthesis from Data-Free Specifications. If in the latter definitions of the problem, one considers specifications defined by RA with no registers, and implementations defined by RT with no registers, then the data in data-words can be ignored and we are in the classical reactive synthesis setting, for which important results are known:

▶ **Theorem 1** ([9]). *Given a (data-free) specification S defined by some (register-free) universal or non-deterministic parity automaton, the realisability problem of S by (register-free) transducers is* ExpTime-c.

4 Unbounded Synthesis

In this section, we consider the unbounded synthesis problem $\textsc{Real}(\mathsf{RA}, \mathsf{RT})$. Thus, we do not fix a priori the number of registers of the implementation. Let us first consider the case of NRA and URA, which are, in our setting, the most natural devices to express data word specifications. By reducing the universality of NRA over finite words (which is undecidable [14]) to our synthesis problems, we show:

▶ **Theorem 2.** $\textsc{Real}(\mathsf{NRA}, \mathsf{RT})$ *is undecidable.*

▶ **Theorem 3.** *For all $k \geq 1$,* $\textsc{Real}(\mathsf{NRA}, \mathsf{RT}[k])$ *is undecidable.*

Now, we can show that the unbounded synthesis problem is also undecidable for URA, answering a question left open in [12].

▶ **Theorem 4.** $\textsc{Real}(\mathsf{URA}, \mathsf{RT})$ *is undecidable.*

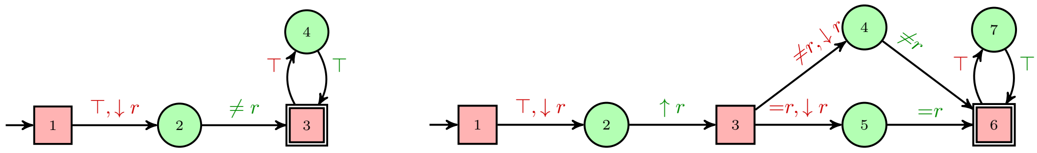
Proof. We present a reduction from the emptiness problem of URA over finite words (which is undecidable by a direct reduction from the universality problem of NRA, undecidable by [14]) to our synthesis problem.

First, consider the relation $S_1 = \{(u\#v, u\#w) \mid u \in \mathsf{DW}_f, v \in \mathsf{DW}$, each data of u appears infinitely often in $w\}$. S_1 is recognised by a 1-register URA which, upon reading a data d in u, stores it in its register and checks that it appears infinitely often in w by visiting a state with maximal parity 2 every time it sees d (all other states have parity 1). Note that for all $k \geq 1$, $S_1 \cap \{(u\#v, u\#w) \mid u \in \mathsf{DW}_f, v, w \in \mathsf{DW}$ and $|\mathsf{dt}(u)| \leq k\}$ is realisable by a k-register transducer: on reading u, store each distinct data in one register, and after the # outputs them in turn in a round-robin fashion. However, S_1 is not realisable: on reading the # separator, any implementation must have all the data of $\mathsf{dt}(u)$ in its registers, but the size of $\mathsf{dt}(u)$ is not bounded (u can have pairwise distinct data and be of arbitary length).

Then, let A be a URA over finite data words. Consider the specification $S = S_1 \cup S_2 \cup T$, where $S_2 = \{(u\#v, u\#w\#(a, \mathsf{d}_0)^\omega) \mid u \in \mathsf{DW}_f, v \in \mathsf{DW}, w \in L(A)\}$ and $T = \{(u, w) \mid u \notin \mathsf{DW}_f\#\mathsf{DW}, w \in \mathsf{DW}\}$. S has total domain, and is recognisable by a URA. Indeed, URA are closed under union, by the same product construction as for the intersection of NRA [10], and each part is URA-recognisable: S_1 is, as described above, S_2 is by simulating A on the output to check $w \in L(A)$ then looping over (a, d_0), and T simply checks a regular property.

Now, if $L(A) \neq \varnothing$, let $w \in L(A)$, and k its number of distinct data. Then S is realisable by a k-register transducer realising S_1 when the number of data in u is lower than or equal to k, and, when it is greater than k, by outputting $u\#\widehat{w}\#(a, \mathsf{d}_0)^\omega$ where $\widehat{w}$ is a membership-preserving renaming of w using k distinct data of u (this can always be done thanks to the so-called "indistinguishability property" stated in [10]). Conversely, if $L(A) = \varnothing$, then S is not realisable. If it were, $S \cap \mathsf{DW}_f\#\mathsf{DW} = S_1$ would be too, as a regular domain restriction, but we have seen above that this is not the case. Thus, S is realisable iff $L(A) = \varnothing$. ◀

However, we show that restricting to DRA allows to recover the decidability, modulo one additional assumption, namely that no output transition of the specification is such that $\mathsf{tst} = \varnothing$ (the output data is different from all register contents). We denote by $\mathsf{DRA}_\varnothing$ this class of DRA. Such assumption rules out pathological, and to our opinion uninteresting and technical cases stemming from the asymmetry between the class of specifications and implementations. E.g., consider the single-register DRA in Fig. 1a (finite labels are arbitrary and not depicted). It starts by reading one input data d and stores it in r, asks that the corresponding output data is different from the content d of r (with $\mathsf{tst} = \varnothing$ depicted here $\neq r$), then accepts any output over any input (transitions $\top$ are always takeable). It is not realisable because transducers necessarily output the content of some register (hence producing a data which already appeared). On the other hand, having tests $\mathsf{tst} = \varnothing$ does not imply unrealisability, as shown by the DRA of Fig. 1b: it starts by reading one data d_1, asks to copy it on the output, then reads another data d_2, and requires that the output is either distinct from d_1 or equal to it, depending on whether $d_2 \neq d_1$. It happens that such specification is realisable by the identity.



(a) An unrealisable DRA.

(b) A similar DRA, suprisingly realisable.

Figure 1 Pathological DRA specifications.

To check for realisability of $\mathsf{DRA}_\varnothing$-specifications with r-registers, we show that it suffices to target transducers with r registers only.

▶ **Proposition 5.** *Let S be a specification defined by a $\mathsf{DRA}_\varnothing$ with r registers. If S is realisable by a register transducer, then it is realisable by a transducer with r registers.*

Proof. Let $S = (\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, \mathcal{D}, Q^S, q_0^S, \delta^S, R^S, c)$ be a $\mathsf{DRA}_\varnothing$ specification realisable by a register transducer $I = (\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, \mathcal{D}, Q^I, q_0^I, \delta^I, R^I)$. Without loss of generality, we assume that S does not conduct assignments on the output and that S is complete. Now, from S, we extract a transducer I' realising S with R^S registers, using I as a guide to make choices for the output. To this end, we simulate I synchronously with S. However, we cannot properly simulate I, since we only have R^S registers, which are used to simulate S. Instead, we keep *constraints* in memory.

Constraints. A constraint represents the equality relations between the registers in R^S and those in R^I (note that such idea is pervasive in the study of registers automata, e.g. to recognise the projection over finite labels). Thus, a constraint is a subset $C \subseteq R^S \times R^I$, which is intended to be a set of equalities between the content of registers in R^S and in R^I. Then, knowing tests $\mathsf{tst}^S, \mathsf{tst}^I$ and assignments asgn^S, asgn^I performed by S and I respectively allows to update the constraints: we define

$$\begin{aligned}\mathrm{next}(C, \mathsf{tst}^S, \mathsf{asgn}^S, \mathsf{tst}^I, \mathsf{asgn}^I) = C \backslash ((\mathsf{asgn}^S \times R^I) \cup (R^S \times \mathsf{asgn}^I))\\ \cup((\mathsf{tst}^S \cup \mathsf{asgn}^S) \times (\mathsf{tst}^I \cup \mathsf{asgn}^I))\end{aligned}$$

For instance, assume $R^I = \{r_1, r_2\}$ and $R^S = \{s_1, s_2\}$, and at some point in a run, we have[3] $C = \{(s_2, r_1), (s_2, r_2)\}$, i.e. $s_2 = r_1 = r_2$ and $s_1 \neq r_1$ (inequalities are implicit, since C is an exhaustive list of equalities). Now, S reads some data d which satisfies the tests $\mathsf{tst}^S = \{s_1\}$ in S and $\mathsf{tst}^I = \varnothing$ in I (such tests are consistent because $s_1 \neq r_1, r_2$), and conducts assignments $\mathsf{asgn}^S = \varnothing$ and $\mathsf{asgn}^I = \{r_2\}$. Then, on the one hand, $s_1 = r_2$ (both contain d), and on the other hand $s_2 = r_1$ (since the content of those registers did not change). Moreover, $r_1 \neq r_2$ since r_2 has been reassigned and $s_1 \neq r_1$ still holds. This is represented by the set of constraints $C' = \{(s_1, r_2), (s_2, r_1)\}$, and indeed, $\mathrm{next}(C, \{s_1\}, \varnothing, \varnothing, \{r_2\}) = C'$.

Abstracting the behaviour of I modifies its language (it somehow simplifies it), but we will see that what we build is still an implementation.

Definition of I'. We build $I' = (\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, \mathcal{D}, Q, q_0, \delta, R^S)$, where $Q = Q^S \times Q^I \times 2^{R^S \times R^I}$ and $q_0 = (q_0^S, q_0^I, R^S \times R^I)$; we now define δ. For each state $(q_{\mathbb{i}}^S, q^I, C) \in Q$, for each input test $(\sigma_{\mathbb{i}}, \mathsf{tst}_{\mathbb{i}}^S) \in \Sigma_{\mathbb{i}} \times \mathsf{Tst}_{R^S}$, we construct a transition $t = (q_{\mathbb{i}}^S, q^I, C) \xrightarrow[I']{\sigma_{\mathbb{i}}, \mathsf{tst}_{\mathbb{i}}^S, \mathsf{asgn}^S, \sigma_{\mathbb{o}}, s_{\mathbb{o}}} ({q'}_{\mathbb{i}}^S, {q'}^I, C')$ whenever there exist the following transitions of S and I:

$$t_{\mathbb{i}}^S = q_{\mathbb{i}}^S \xrightarrow[S]{\sigma_{\mathbb{i}}, \mathsf{tst}_{\mathbb{i}}^S, \mathsf{asgn}^S} q_{\mathbb{o}}^S \qquad t_{\mathbb{o}}^S = q_{\mathbb{o}}^S \xrightarrow[S]{\sigma_{\mathbb{o}}, \mathsf{tst}_{\mathbb{o}}^S} {q'}_{\mathbb{i}}^S \qquad t^I = q^I \xrightarrow[I]{\sigma_{\mathbb{i}}, \mathsf{tst}^I, \mathsf{asgn}^I, \sigma_{\mathbb{o}}, r_{\mathbb{o}}} {q'}^I$$

such that, for some fixed arbitrary order on R^S, we have:

(i) $\mathsf{tst}^I = \{r \in R^I \mid \exists s \in \mathsf{tst}_{\mathbb{i}}^S, (s, r) \in C\}$
(ii) $C' = \mathrm{next}(C, \mathsf{tst}_{\mathbb{i}}^S, \mathsf{asgn}^S, \mathsf{tst}^I, \mathsf{asgn}^I)$
(iii) $\mathsf{tst}_{\mathbb{o}}^S = \{s \in R^S \mid (s, r_{\mathbb{o}}) \in C'\}$
(iv) $s_{\mathbb{o}} = \min \mathsf{tst}_{\mathbb{o}}^S$

[3] For readability, we confuse a register with its content.

Item (i) ensures with the help of constraints that in any reachable configuration of I', there exists at least one input data which satisfies both $\mathsf{tst}_{\mathbb{i}}^S$ and tst^I, which allows I' to synchronise S with I. Note that this does not means that I' is the synchronous product of S and I *on any input*: since I' only has the registers of S, it cannot discriminate data as subtly as I, and might thus adopt a different behaviour. For instance, it can be that upon reading some input data word, at some point, I would store some input data d in some register r that S would not, and use it later on in a test $\mathsf{tst}^I = \{r\}$ to take different actions, while neither I' nor S could discriminate between those choices: on reading d, I' simulates S with $\mathsf{tst}_{\mathbb{i}}^S = \varnothing$ and synchronously simulates in I the transition with input test $\mathsf{tst}^I = \varnothing$. Nevertheless, we show the *existence* of some relational data word common to I and I' for each run of I' (which is also a run of S). This is sufficient to conclude that I' realises S, because then each run of I', interpreted as a run of S, is accepting. Then, items (iii) and (iv) ensures the same property as item (i) does, but this time on output positions.

We shall see that for a transition t such that (q^S, q^I, C) is accessible on some finite input data word, $t_{\mathbb{i}}^S$, $t_{\mathbb{o}}^S$ and t^I exist and are unique. So, for a run $\rho = t_1 t_2 \dots$ of I', we define $\rho^S = t_{\mathbb{i}1}^S t_{\mathbb{o}1}^S t_{\mathbb{i}2}^S t_{\mathbb{o}2}^S \dots$ and $\rho^I = t_1^I t_2^I \dots$

Proof of Correctness. Let us show that I' is indeed a transducer realising S: we show that for all $u_{\mathbb{i}} \in \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})$, there exists a unique sequence of transitions ρ in I', a unique output data word $u_{\mathbb{o}} \in \mathsf{DW}(\Sigma_{\mathbb{o}}, \mathcal{D})$ (we denote $u = \langle u_{\mathbb{i}}, u_{\mathbb{o}} \rangle$) and a $w \in L(I)$ such that:

1. ρ^I is the run of I over w
2. ρ^S is the run of S over w
3. ρ is the run of I' over u
4. ρ^S is the run of S over u

Note that the above properties imply $\mathsf{lab}(u) = \mathsf{lab}(w)$, but it can be that $u \neq w$, which is consistent with the observations we made. Let us show that they entail the result we need: let $u_{\mathbb{i}} \in \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})$ be some input data word. By property 3, $f_{I'}(u_{\mathbb{i}})$ exists and is unique, so I' has total domain. Now, by denoting ρ^S the run of S over $u = \langle u_{\mathbb{i}}, f_{I'}(u_{\mathbb{i}}) \rangle$, we know by property 2 that there exists $w \in L(I)$ such that ρ^S is the run of S over w. Then, ρ^S is accepting because I realises S so $w \in L(S)$, hence $u \in L(S)$. Thus, I' realises S.

The proof of properties 1-4 is rather technical, and can be found in [7]. ◀

Such result allows us to reduce unbounded synthesis to bounded synthesis for $\mathsf{DRA}_{\varnothing}$. Bounded synthesis is in ExpTime for DRA (Thm. 13) and is the topic of the next section.

▶ **Theorem 6.** Real$(\mathsf{DRA}_{\varnothing}, \mathsf{RT})$ *is decidable in* ExpTime*.*

5 Bounded Synthesis: A Generic Approach

In this section, we study the setting where target implementations are register transducers in the class $\mathsf{RT}[k]$, for some $k \geq 0$ that we now fix for the whole section. For the complexity analysis, we assume k is given as input, in unary. Indeed, describing a k-register automaton in general requires $O(k)$ bits, and not $O(\log k)$ bits. We prove the decidable cases of the first line of Table 1 (page 3), by reducing the problems to realisability problems for data-free specifications.

Abstract Actions. We reduce the problem to a finite alphabet problem. Since we synthesise k-register transducers, we take the input and output actions of the transducers as symbols of our finite input and output alphabets. Let $R_k = \{1, \dots, k\}$ and $\mathsf{Tst}_k = \mathsf{Asgn}_k = 2^{R_k}$. The finite input actions are $A^k_{\mathbb{i}} = \Sigma_{\mathbb{i}} \times \mathsf{Tst}_k$ which corresponds to picking a label and a test over the k registers, and the output actions are $A^k_{\mathbb{o}} = \Sigma_{\mathbb{o}} \times \mathsf{Asgn}_k \times R_k$, corresponding to picking some output symbol, some assignment and some register whose content is to be output.

An alternating sequence of actions $\overline{a} = (\sigma^1_{\mathbb{i}}, \mathsf{test}_1)(\sigma^1_{\mathbb{o}}, \mathsf{asgn}_1, r_1) \dots \in (A^k_{\mathbb{i}} A^k_{\mathbb{o}})^\omega$ abstracts a set of relational data words of the form $w = (\sigma^1_{\mathbb{i}}, d^1_{\mathbb{i}})(\sigma^1_{\mathbb{o}}, d^1_{\mathbb{o}}) \dots \in \mathsf{RW}(\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, \mathcal{D})$ via a compatibility relation that we now define. We say that w is *compatible* with $\overline{a}$ if there exists a sequence of register configurations $\tau_0\tau_1 \dots \in (R_k \to \mathcal{D})^\omega$ such that $\tau_0 = \tau_0^{R_k}$ and for all $i \geq 1$, $\tau_i, d^i_{\mathbb{i}} \models \mathsf{test}_i$, $d^i_{\mathbb{o}} = \tau_i(r_i)$ and $\tau_{i+1} = \mathrm{next}(\tau_i, d^i_{\mathbb{i}}, \mathsf{asgn}_i)$. Note that this sequence is unique if it exists. We denote by $\mathsf{Comp}(\overline{a})$ the set of relational data words compatible with $\overline{a}$. Given a specification S, we let $W_{S,k} = \{\overline{a} \mid \mathsf{Comp}(\overline{a}) \subseteq S\}$. The set $W_{S,k}$ can be seen as a specification over the finite input (resp. output) alphabets $A^k_{\mathbb{i}}$ (resp. $A^k_{\mathbb{o}}$).

▶ **Theorem 7** (Transfer). *Let S be a data word specification. The following are equivalent:*

1. *S is realisable by a transducer with k registers.*
2. *The (data-free) word specification $W_{S,k}$ is realisable by a (register-free) finite transducer.*

Proof. Let T be a transducer with k registers realising S. The transducer T can be seen as a finite transducer T' over input alphabet $A^k_{\mathbb{i}}$ and output alphabet $A^k_{\mathbb{o}}$. Indeed, since the transition function of T is total, it is also the case of T' (this is required by the definition of transducer defining implementations).

Let us show that $W_{S,k}$ is realisable by T', i.e. $L(T') \subseteq W_{S,k}$. Take a sequence $\overline{a} = a_1e_1a_2e_2 \dots \in L(T')$. We show that $\mathsf{Comp}(\overline{a}) \subseteq S$. Let $w \in \mathsf{Comp}(\overline{a})$. Then, there exists a run $q_0q_1q_2 \dots$ of T' on $\overline{a}$ since $\overline{a} \in L(T')$. By definition of compatibility for w, there exists a sequence of register configurations $\tau_0\tau_1 \dots \in (R_k \to \mathcal{D})^\omega$ satisfying the conditions in the definition of compatibility. From this we can deduce that $(q_0, \tau_0)(q_1, \tau_1) \dots$ is an initial sequence of configurations of T over w, so $w \in L(T)$. Finally, $L(T) \subseteq S$, since T realises S.

Conversely, suppose that $W_{S,k}$ is realisable by some finite transducer T' over the input (output) alphabets $A^k_{\mathbb{i}}$ $(A^k_{\mathbb{o}})$. Again, the transducer T can be seen as a transducer with k registers over data words. We show that T realises S, i.e., $L(T) \subseteq S$. Let $w \in L(T)$. The run of T over w induces a sequence of actions $\overline{a}$ in $(A^k_{\mathbb{i}} A^k_{\mathbb{o}})^\omega$ which, by definition of compatibility, satisfies $w \in \mathsf{Comp}(\overline{a})$. Moreover, $\overline{a} \in L(T')$. Hence, since T' realises $W_{S,k}$, we get $\mathsf{Comp}(\overline{a}) \subseteq S$, so $w \in S$, concluding the proof. ◀

5.1 The case of URA specifications

In this section, we show that for any S a data word specification given as some URA, the language $W_{S,k}$ is effectively ω-regular, entailing the decidability of Real(URA, RT[k]), by Theorem 7 and the decidability of (data-free) synthesis. Let us first prove a series of intermediate lemmas.

We define an operation $\otimes$ between relational data words $w \in \mathsf{RW}(\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, \mathcal{D})$ and sequences of actions $\overline{a} \in (A^k_{\mathbb{i}} A^k_{\mathbb{o}})^\omega$ as follows: $w \otimes \overline{a} \in \mathsf{RW}(A^k_{\mathbb{i}}, A^k_{\mathbb{o}}, \mathcal{D})$ is defined only if for all $i \geq 1$, $\mathsf{lab}(w[i]) = \mathsf{lab}(\overline{a}[i])$ where $\mathsf{lab}(\overline{a}[i])$ is the first component of $\overline{a}[i]$ (a label in $\Sigma_{\mathbb{i}} \cup \Sigma_{\mathbb{o}}$), by $(w \otimes \overline{a})[i] = (\overline{a}[i], \mathsf{dt}(w[i]))$.

▶ **Lemma 8.** *The language $L_k = \{w \otimes \overline{a} \mid w \in \mathit{Comp}(\overline{a})\}$ is definable by some* NRA.

Proof. We define an NRA with k registers which roughly follows the actions it reads on its input. Its set of states is $\{q\} \cup \mathsf{Asgn}_R$, with initial state q. In state q, it is only allowed to read labelled data in $A^k_{\mathbb{i}} \times \mathcal{D}$. On reading $(\sigma_{\mathbb{i}}, \mathsf{tst}, d)$, it guesses some assignment asgn, performs the test tst and the assignment asgn and goes to state asgn. In any state $\mathsf{asgn} \in \mathsf{Asgn}_R$, it is only allowed to read labelled data of the form $(\sigma_{\mathbb{O}}, \mathsf{asgn}, r, d)$, for which it tests whether d is equal to the content of r. It does no assignment and moves back to state q. All states are accepting (i.e. have maximal even parity 2). Such NRA has size $O(2^{k^2})$. ◀

Let S be a specification defined by some URA A_S with set of states Q. The following subset of L_k is definable by some NRA, where $\overline{S}$ denotes the complement of S:

▶ **Lemma 9.** *The language $L_{\overline{S},k} = \{w \otimes \overline{a} \mid w \in \mathit{Comp}(\overline{a}) \cap \overline{S}\}$ is definable by some* NRA.

Proof. Since S is definable by the URA A_S, $\overline{S}$ is NRA-definable with the same automaton, denoted now $A_{\overline{S}}$, interpreted as an NRA. Let B be some NRA defining L_k (it exists by Lemma 8). It now suffices to take a product of $A_{\overline{S}}$ and B to get an NRA defining $L_{\overline{S},k}$. ◀

Given a data word language L, we denote by $\mathsf{lab}(L) = \{\mathsf{lab}(w) \mid w \in L\}$ its projection on labels. The language $W_{S,k}$ is obtained as the complement of the label projection of $L_{\overline{S},k}$:

▶ **Lemma 10.** $W_{S,k} = \overline{\mathit{lab}(L_{\overline{S},k})}$.

Proof. Let $\overline{a} \in (A_{\mathbb{i}}^k A_{\mathbb{O}}^k)^\omega$. Then, $\overline{a} \notin W_{S,k} \Leftrightarrow \mathsf{Comp}(\overline{a}) \not\subseteq S \Leftrightarrow \exists w \in \mathsf{RW}, w \in \mathsf{Comp}(\overline{a}) \cap \overline{S} \Leftrightarrow \exists w \in \mathsf{RW}, w \otimes \overline{a} \in L_{\overline{S},k} \Leftrightarrow \overline{a} \in \mathsf{lab}(L_{\overline{S},k})$. ◀

We are now able to show regularity of $W_{S,k}$.

▶ **Lemma 11.** *Let S be a data word specification, $k \geq 0$. If S is definable by some* URA *with n states and r registers, then $W_{S,k}$ is effectively ω-regular, definable some deterministic parity automaton with $O(2^{n^2 16^{(r+k)^2}})$ states and $O(n.4^{(r+k)^2})$ priorities.*

Proof. First, $L_{\overline{S},k}$ is definable by some NRA with $O(2^{k^2} n)$ states and $O(r+k)$ registers by Lemma 10, obtained as product between the NRA $A_{\overline{S}}$ and the automaton obtained in Lemma 8, of size $O(2^{k^2})$. It is known that the projection on the alphabet of labels of a language of data words recognised by some NRA is effectively regular [10]. The same construction, which is based on extending the state space with register equality types, carries over to ω-words, and one obtains a non-deterministic parity automaton with $O(n.4^{(r+k)^2})$ states and d priorities recognising $\mathsf{lab}(L_{\overline{S},k})$. It can be complemented into a deterministic parity automaton with $O(2^{n^2.16^{(r+k)^2}})$ states and $O(n.4^{(r+k)^2})$ priorities using standard constructions [15]. ◀

We are now able to reprove the following result, known from [12]:

▶ **Theorem 12.** *For all $k \geq 0$,* Real(URA, RT[k]) *is in* 2ExpTime.

Proof. By Lemma 11, we construct a deterministic parity automaton $P_{S,k}$ for $W_{S,k}$. Then, according to Theorem 7, it suffices to check whether it is realisable by a (register-free) transducer. This is decidable by Theorem 1. The way to decide it is to see $P_{S,k}$ as a two-player parity game and check whether the protagonist has a solution. Parity games can be solved in time $O(m^{\log d})$ [3] where m is the number of states of the game and d the number of priorities. Overall, solving it requires doubly exponential time, more precisely in $O(2^{n^3 16^{(r+k)^2}})$. ◀

In [11], it is shown that the complexity of the problem is actually only singly exponential in k, and such analysis extends to our construction. Similarly, when the specification automaton is deterministic, we can show that:

▶ **Theorem 13.** Real(DRA, RT[k]) *is in* ExpTime.

5.2 The case of test-free NRA specifications

Unfortunately, by Theorem 3, the synthesis problem for specifications expressed as NRA is undecidable, even when the number of registers of the implementation is bounded. And indeed, if we mimic the reasoning of the previous section, Lemma 10 does not allow to conclude because $L_{\overline{S},k}$ is definable by a URA but the string projection of a URA is not ω-regular in general. E.g., consider $L = \{(r, d_1) \dots (r, d_n)(g, d'_1) \dots (g, d'_m) \mid \forall i \neq j, d_i \neq d_j \wedge \forall 1 \leq i \leq n, \exists j, d'_j = d_i\}$, which consists in a word $w \in r^n$ with pairwise distinct data followed by a word $w \in g^m$ which contains at least all the data of w (it is over finite words for simplicity but can be extended to ω-words). L is recognised by the URA which, on reading (r, d_i), universally triggers a run checking three properties: firstly, once a label g is read, only g's are read, secondly, (r, d_i) does not appear again, and thirdly, (g, d_i) appears at least once. Now, it is readily seen that $\mathsf{lab}(L) = \{r^n g^m \mid m \geq n\}$, which is not regular. Here, we defined L over finite words for simplicity but it is easily extended to an ω-language which is not ω-regular.

Thus, in this section, we consider a restriction on NRA which do not perform tests on input data. A *test-free register automaton* is a tuple $A = (\Sigma_{\mathbb{i}}, \Sigma_{\mathbb{o}}, \mathcal{D}, Q, q_0, \delta, R, c)$ such that $\delta \subseteq Q \times \Sigma_{\mathbb{i}} \times \mathsf{Asgn}_R \times \Sigma_{\mathbb{o}} \times R \times Q$. Such a register automaton reads two labelled data at once. In a configuration (q, τ), when reading $(\sigma_{\mathbb{i}}, d_{\mathbb{i}})(\sigma_{\mathbb{o}}, d_{\mathbb{o}})$, it can fire any transition of the form $(q, \sigma_{\mathbb{i}}, \mathsf{asgn}, r, \sigma_{\mathbb{o}}, q') \in \delta$ such that $\tau(r) = d_{\mathbb{o}}$ and move to configuration (q', τ') where $\tau' = \mathrm{next}(\tau, \mathsf{asgn}, d_{\mathbb{i}})$. It is easily seen that a test-free register automaton can be converted into a proper register automaton, justifying its name. Such automata will be interpreted by a non-deterministic parity acceptance condition; we denote the class $\mathsf{NRA}_{\mathsf{tf}}$[4].

It is not clear whether $W_{S,k}$ is regular for such specifications, but we show that it suffices to consider another set denoted $W^{\mathsf{tf}}_{S,k}$ which is easier to analyse (and can be proven regular), and based on the behaviour of S over input with pairwise distinct data. The intuition behind restricting to such case is that $\mathsf{NRA}_{\mathsf{tf}}$ cannot conduct test on input data, so they behave the same on an input word whose data are all distinct, and such choice ensures that two equal input data will not ease the task of the implementation. An interesting side-product of this approach is that it implies that we can restrict to test-free implementations. A test-free transducer is a transducer whose transitions do not depend on tests over input data; formally $\delta : Q \times \Sigma_{\mathbb{i}} \to \mathsf{Asgn}_R \times \Sigma_{\mathbb{o}} \times R \times Q$. In the following, we let $\textsc{AllDiff}$ denote the set of relational data words whose input data are pairwise distinct: $\textsc{AllDiff} = \{w = (\sigma^1_{\mathbb{i}}, d^1_{\mathbb{i}})(\sigma^1_{\mathbb{o}}, d^1_{\mathbb{o}}) \dots \in \mathsf{RW} \mid \forall 0 \leq i < i', d^i_{\mathbb{i}} \neq d^{i'}_{\mathbb{i}}\}$; by convention $d^0_{\mathbb{i}} = \mathsf{d}_0$.

▶ **Proposition 14.** *Let S be a $\mathsf{NRA}_{\mathsf{tf}}$ specification. The following are equivalent:*

(i) *S is realisable*

(ii) *$W^{\mathsf{tf}}_{S,k} = \{\overline{a} \in (A^{\varnothing}_{\mathbb{i}} A^k_{\mathbb{o}})^\omega \mid \mathsf{Comp}(\overline{a}) \cap S \cap \textsc{AllDiff} \neq \varnothing\}$, where $A^{\varnothing}_{\mathbb{i}} = \Sigma_{\mathbb{i}} \times \{\varnothing\}$, has domain $(A^{\varnothing}_{\mathbb{i}})^\omega$ and is realisable (by a register-free transducer)*

(iii) *S is realisable by a test-free transducer*

Proof. (i) $\Rightarrow$ (ii): If S is realisable, then by Theorem 7 $W_{S,k}$ has total domain and is realisable by some transducer I. Now, since transducers are closed under regular domain restriction, $W^{\varnothing}_{S,k} = W_{S,k} \cap (A^{\varnothing}_{\mathbb{i}} A^k_{\mathbb{o}})^\omega$ has domain $(A^{\varnothing}_{\mathbb{i}})^\omega$ and is realisable by $I \cap (A^{\varnothing}_{\mathbb{i}} A^k_{\mathbb{o}})^\omega$. Moreover, $W^{\varnothing}_{S,k} \subseteq W^{\mathsf{tf}}_{S,k}$. Indeed, if $\mathsf{Comp}(\overline{a}) \subseteq S$, then, since S has total domain and $\overline{a} \in (A^{\varnothing}_{\mathbb{i}} A^k_{\mathbb{o}})^\omega$, $\mathsf{Comp}(\overline{a}) \cap S \cap \textsc{AllDiff} \neq \varnothing$. Thus, $W^{\mathsf{tf}}_{S,k}$ also has domain $(A^{\varnothing}_{\mathbb{i}})^\omega$ and is realisable by any transducer realising $W^{\varnothing}_{S,k}$.

[4] The bounded synthesis of URA is already decidable, so we do not consider their test-free restriction.

(iii) ⇒ (i): is trivial.

(ii) ⇒ (iii): Intuitively, $\mathsf{NRA}_{\mathsf{tf}}$ can only rearrange input data (duplicate, erase, copy) regardless of the actual data values (as there are no tests), so its behaviour on AllDiff determines its behaviour on the entire domain. This can be formalised precisely through a notion of data origins given a run (this notion is also made explicit in [5]).

To a run $\rho = q_0 \xrightarrow{\sigma_{\mathbb{i}}^1, \mathsf{asgn}^1, r^1, \sigma_{\mathbb{o}}^1} q_1 \xrightarrow{\sigma_{\mathbb{i}}^2, \mathsf{asgn}^2, r^2, \sigma_{\mathbb{o}}^2} q_2 \dots$ corresponds the origin function $o_\rho : j \mapsto \max\{i \leq j \mid r_j \in \mathsf{asgn}_i\}$, with the convention $\max \varnothing = 0$.

Now, for an origin function $o : \mathbb{N}\backslash\{0\} \to \mathbb{N}$ and for a relational data word $w \in \mathsf{RW}$, we say w is compatible with the origin function o, denoted $w \models o$, whenever for all $j \geq 1$, $\mathsf{dt}(\mathsf{out}(w)[j]) = \mathsf{dt}(\mathsf{inp}(w)[o(j)])$, with the convention $\mathsf{dt}(\mathsf{inp}(w)[0]) = \mathsf{d}_0$.

The following lemma shows that actual data values in a word w do not matter with respect to membership in some $\mathsf{NRA}_{\mathsf{tf}}$, only the compatibility with origin functions does:

▶ **Lemma 15.** *Let $w \in RW$ and ρ a sequence of transitions of some $\mathsf{NRA}_{\mathsf{tf}}$. Then,*

(i) *If ρ is a run over w, then $w \models o_\rho$.*

(ii) *If ρ is a run over w and $w \in$ AllDiff, then for all $o : \mathbb{N}\backslash\{0\} \to \mathbb{N}$, $w \models o \Leftrightarrow o = o_\rho$.*

(iii) *If w and ρ have the same finite labels and if $w \models o_\rho$, then ρ is a run over w.*

Proof. (i) and (iii) follow from the semantics of $\mathsf{NRA}_{\mathsf{tf}}$, which do not conduct any test on the input data. The ⇐ direction of (ii) is exactly (i). Now, assume $w \in$ AllDiff admits ρ as a run, and let o such that $w \models o$. Then, let $j \geq 1$ be such that $\mathsf{dt}(\mathsf{out}(w)[j]) = \mathsf{dt}(\mathsf{inp}(w)[o(j)])$. By (i) we know that $\mathsf{dt}(\mathsf{out}(w)[j]) = \mathsf{dt}(\mathsf{inp}(w)[o_\rho(j)])$, so $\mathsf{dt}(\mathsf{inp}(w)[o(j)]) = \mathsf{dt}(\mathsf{inp}(w)[o_\rho(j)])$. Since $w \in$ AllDiff, this implies $o(j) = o_\rho(j)$, so, overall, $o = o_\rho$. ◀

Now, assume $W^{\mathsf{tf}}_{S,k}$ is realisable by some transducer I. We show that I, when ignoring the $\varnothing$ input tests, is actually an implementation of S. Thus, let I' be the same transducer as I except that all input transitions $(\sigma_{\mathbb{i}}, \varnothing)$ are now simply labelled $\sigma_{\mathbb{i}}$. Note that I', interpreted as a register transducer, is test-free. Let $w \in \mathsf{DW}$, and $\overline{a}_{\mathbb{i}} = \mathsf{lab}(w) \times \varnothing^\omega$ be the input action in $A^\varnothing_{\mathbb{i}}$ with same finite labels as w. Let $\overline{a} = I(\overline{a}_{\mathbb{i}})$, and let $w' \in \mathsf{Comp}(\overline{a}) \cap S \cap$ AllDiff (such w' exists because $W^{\mathsf{tf}}_{S,k}$ has domain $(A^\varnothing_{\mathbb{i}})^\omega$ and I realises $W^{\mathsf{tf}}_{S,k}$). Then, since $\mathsf{lab}(w) = \mathsf{lab}(w')$, they admit the same run ρ^I in I, so $w, w' \models o_{\rho^I}$. Then, $w' \in S$, so it admits an accepting run ρ^S in S, which implies $w' \models o_{\rho^S}$. Moreover, $w' \in$ AllDiff so, by Lemma 15 ii, we get $o_{\rho^I} = o_{\rho^S}$. Therefore, $w \models o_{\rho^S}$, so, by iii, w admits ρ^S as a run, i.e. $w \in S$. Overall, $L(I) \subseteq S$ meaning that I is a (test-free) implementation of S. *End of proof of Prop. 14* ◀

Finally, $W^{\mathsf{tf}}_{S,k} = \{\overline{a} \in (A^\varnothing_{\mathbb{i}} A^k_{\mathbb{o}})^\omega \mid \mathsf{Comp}(\overline{a}) \cap S \cap \text{AllDiff} \neq \varnothing\}$ is regular. Indeed, $W^{\mathsf{tf}}_{S,k} = \{\overline{a} \in (A^\varnothing_{\mathbb{i}} A^k_{\mathbb{o}})^\omega \mid \mathsf{Comp}(\overline{a}) \cap S^\varnothing \neq \varnothing\}$, where $S^\varnothing$ is the same automaton as S except that all transitions $q \xrightarrow{\sigma_{\mathbb{i}}, \mathsf{asgn}, r, \sigma_{\mathbb{o}}} q'$ have been replaced with $q \xrightarrow{\sigma_{\mathbb{i}}, \varnothing, \mathsf{asgn}, r, \sigma_{\mathbb{o}}} q'$, because, for all $\overline{a} \in (A^\varnothing_{\mathbb{i}} A^k_{\mathbb{o}})^\omega$, $\mathsf{Comp}(\overline{a}) \cap S \cap \text{AllDiff} \neq \varnothing \Leftrightarrow \mathsf{Comp}(\overline{a}) \cap S^\varnothing \neq \varnothing$ (the ⇒ direction is trivial, and the ⇐ stems from the fact that an AllDiff input only takes $\mathsf{tst} = \varnothing$ transitions).

Then, $L^{\mathsf{tf}}_{S,k} = \{w \otimes \overline{a} \in \mathsf{RW} \otimes (A^\varnothing_{\mathbb{i}} A^k_{\mathbb{o}})^\omega \mid w \in \mathsf{Comp}(\overline{a}) \cap S^\varnothing\}$ is NRA-definable. Indeed, S is $\mathsf{NRA}_{\mathsf{tf}}$-definable, so $S^\varnothing$ is NRA-definable, and by Lemma 8, $L_k = \{w \otimes \overline{a} \mid w \in \mathsf{Comp}(\overline{a})\}$ is NRA-definable, so their product recognises $L^{\mathsf{tf}}_{S,k}$. Finally, $W^{\mathsf{tf}}_{S,k} = \mathsf{lab}(L^{\mathsf{tf}}_{S,k})$, and the projection of a NRA over some finite alphabet is regular [10].

Overall, by Theorems 1 and 7, we get (the complexity analysis is the same as for URA):

▶ **Theorem 16.** *For all $k \geq 0$, Real$(\mathsf{NRA}_{\mathsf{tf}}, \mathsf{RT}[k])$ is decidable and in 2ExpTime.*

6 Synthesis and Uniformisation

In this section, we discuss the relation between realisability and uniformisation of relations: in this paper, if S is realisable by a register transducer, then, in particular, it has universal domain, i.e. $\mathsf{inp}(S) = \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})$, otherwise it cannot be that $L(T) \subseteq S$ for T a register transducer, since by definition $\mathsf{inp}(T) = \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})$. However, when defining a specification, the user might be interested only in a subset of behaviours (for instance, s/he knows that all input data will be pairwise distinct). In the finite alphabet setting, since the formalisms used to express specifications are closed under complement (whether it is LTL or ω-automata), it suffices to complete the specification by allowing any behaviour on the input not considered. However, since register automata are not closed under complement, such approach is not sufficient here. Thus, it is relevant to generalise the realisability problem to the case where the domain of the specification is not universal. This can be done by equipping register transducers with an acceptance condition. It is also necessary to adapt the notion of realisability; otherwise, any transducer accepting no words realises any specification. A natural way is to consider synthesis as an instance of the uniformisation problem. An (implementation) function $f : I \to O$ is said to *uniformise* a (specification) relation $R \subseteq I \times O$ whenever $\mathrm{dom}(f) = \mathrm{dom}(R)$ and for all $i \in \mathrm{dom}(f), (i, f(i)) \in R$. In the context of reactive synthesis, where $f = f_I$ is defined from an implementation I and R is given as a language of relational words, it can be rephrased as $\mathsf{inp}(L(I)) = \mathsf{inp}(R)$ and for all $w_{\mathbb{i}} \in \mathsf{inp}(L(I)), \langle w_{\mathbb{i}}, f_I(w_{\mathbb{i}})\rangle \in R$. Note that such definition coincides with the one of realisability when the class of implementations has universal domain. In the following, we denote by $\textsc{Unif}(\mathcal{S}, \mathcal{I})$ the uniformisation problem from specifications in $\mathcal{S}$ to implementations in $\mathcal{I}$. Unfortunately, this setting is actually much harder, as shown by the next two theorems:

▶ **Theorem 17.** *Given S a specification represented by a* DRA*, checking whether $\mathsf{inp}(S) = \mathsf{DW}(\Sigma_{\mathbb{i}}, \mathcal{D})$ is undecidable.*

While the uniformisation setting obviously preserves the undecidability results from the synthesis setting, the above result allows to show that the somehow more general uniformisation problem is undecidable. For instance, we can prove:

▶ **Theorem 18.** *For all $k \geq 1$, $\textsc{Unif}(\mathsf{URA}, \mathsf{RT}[k])$ is undecidable.*

If the domain is DRA-recognisable, it is possible to reduce the uniformisation problem to realisability, by allowing any behaviour on the complement on the domain (which is DRA-recognisable). However, such property is undecidable as a direct corollary of Theorem 17.

7 Conclusion

In this paper, we have given a picture of the decidability landscape of the synthesis of register transducers from register automata specifications. We studied the parity acceptance condition because of its generality, but our results allow to reduce the synthesis problem for register automata specifications to the one for finite automata while preserving the acceptance condition. We have also introduced and studied test-free NRA, which do not have the ability to test their input, but still have the power of duplicating, removing or copying the input data to form the output. We have shown that they allow to recover decidability in presence of non-determinism, in the bounded case. We leave open the unbounded case, which we conjecture to be decidable. As future work, we want to study synthesis problems for specifications given by logical formulae, for decidable data words logics such as two-variable fragments of FO [2, 17, 4].

References

1 Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. *Graph Games and Reactive Synthesis*, pages 921–962. Springer International Publishing, Cham, 2018.

2 Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, Luc Segoufin, and Claire David. Two-Variable Logic on Words with Data. In *Proceedings of the 21th IEEE Symposium on Logic in Computer Science (LICS 2006)*, pages 7–16. ACM, 2006. doi:10.1109/LICS.2006.51.

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 (STOC 2017)*, pages 252–263. ACM, 2017. doi:10.1145/3055399.3055409.

4 Luc Dartois, Emmanuel Filiot, and Nathan Lhote. Logics for Word Transductions with Synthesis. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 295–304. ACM, 2018. doi:10.1145/3209108.3209181.

5 Antoine Durand-Gasselin and Peter Habermehl. Regular Transformations of Data Words Through Origin Information. In *Proceedings of the 19th International Conference on Foundations of Software Science and Computation Structures (FOSSACS 2016)*, volume 9634 of *Lecture Notes in Computer Science*, pages 285–300. Springer, 2016. doi:10.1007/978-3-662-49630-5_17.

6 Rüdiger Ehlers, Sanjit A. Seshia, and Hadas Kress-Gazit. Synthesis with Identifiers. In *Proceedings of the 15th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2014)*, volume 8318 of *Lecture Notes in Computer Science*, pages 415–433. Springer, 2014. doi:10.1007/978-3-642-54013-4_23.

7 Léo Exibard, Emmanuel Filiot, and Pierre-Alain Reynier. Synthesis of Data Word Transducers. *CoRR*, abs/1905.03538, 2019. arXiv:1905.03538.

8 Diego Figueira and M. Praveen. Playing with Repetitions in Data Words Using Energy Games. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2018)*, pages 404–413. ACM, 2018. doi:10.1145/3209108.3209154.

9 J.R. Büchi and L.H. Landweber. Solving Sequential Conditions by Finite-State Strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

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

11 Ayrat Khalimov and Orna Kupferman. Register Bounded Synthesis. In *Proceedings of the 30th International Conference on Concurrency Theory (CONCUR 2019)*, 2019. To appear.

12 Ayrat Khalimov, Benedikt Maderbacher, and Roderick Bloem. Bounded Synthesis of Register Transducers. In *Proceedings of the 16th International Symposium on Automated Technology for Verification and Analysis (ATVA 2018)*, volume 11138 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2018.

13 Andrzej S. Murawski, Steven J. Ramsay, and Nikos Tzevelekos. Bisimilarity in Fresh-Register Automata. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS 2015)*, pages 156–167. IEEE Computer Society, 2015. doi:10.1109/LICS.2015.24.

14 Frank Neven, Thomas Schwentick, and Victor Vianu. Finite State Machines for Strings over Infinite Alphabets. *ACM Trans. Comput. Logic*, 5(3):403–435, 2004. doi:10.1145/1013560.1013562.

15 Nir Piterman. From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata. *Logical Methods in Computer Science*, 3(3), 2007.

16 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *ACM Symposium on Principles of Programming Languages, POPL*. ACM, 1989.

17 Thomas Schwentick and Thomas Zeume. Two-Variable Logic with Two Order Relations. *Logical Methods in Computer Science*, 8(1), 2012. doi:10.2168/LMCS-8(1:15)2012.

18 Luc Segoufin. Automata and Logics for Words and Trees over an Infinite Alphabet. In *Proceedings of the 15th Annual Conference of the EACSL on Computer Science Logic (CSL 2006)*, volume 4207 of *Lecture Notes in Computer Science*, pages 41–57. Springer, 2006. doi:10.1007/11874683_3.

Register-Bounded Synthesis

Ayrat Khalimov
School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel
ayrat.khalimov@gmail.com

Orna Kupferman
School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel
orna@cs.huji.ac.il

Abstract

Traditional *synthesis* algorithms return, given a specification over finite sets of input and output Boolean variables, a finite-state transducer all whose computations satisfy the specification. Many real-life systems have an infinite state space. In particular, behaviors of systems with a finite control yet variables that range over infinite domains, are specified by automata with infinite alphabets. A *register automaton* has a finite set of registers, and its transitions are based on a comparison of the letters in the input with these stored in its registers. Unfortunately, reasoning about register automata is complex. In particular, the synthesis problem for specifications given by register automata, where the goal is to generate correct register transducers, is undecidable.

We study the synthesis problem for systems with a bounded number of registers. Formally, the *register-bounded realizability problem* is to decide, given a specification register automaton A over infinite input and output alphabets and numbers k_s and k_e of registers, whether there is a system transducer T with at most k_s registers such that for all environment transducers T' with at most k_e registers, the computation $T \| T'$, generated by the interaction of T with T', satisfies the specification A. The *register-bounded synthesis problem* is to construct such a transducer T, if exists. The bounded setting captures better real-life scenarios where bounds on the systems and/or its environment are known. In addition, the bounds are the key to new synthesis algorithms, and, as recently shown in [24], they lead to decidability. Our contributions include a stronger specification formalism (universal register parity automata), simpler algorithms, which enable a clean complexity analysis, a study of settings in which both the system and the environment are bounded, and a study of the theoretical aspects of the setting; in particular, the differences among a fixed, finite, and infinite number of registers, and the *determinacy* of the corresponding games.

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

Keywords and phrases Synthesis, Register Automata, Register Transducers

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.25

Related Version A full version of the paper is available at `https://www.cs.huji.ac.il/~ornak/publications/concur19.pdf`.

1 Introduction

Synthesis is the automated construction of a system from its specification. The specification distinguishes between outputs, generated by the system, and inputs, generated by its environment. The system should *realize* the specification, namely satisfy it against all possible environments. Thus, for every sequence of inputs, the system should generate a sequence of outputs so that the induced computation satisfies the specification [10, 30]. The systems are modelled by *transducers*: automata whose transitions are labeled by letters from the input alphabet, which trigger the transition, and letters from the output alphabet, which are generated when the transition is taken. Since its introduction, synthesis has been one of the most studied problems in formal methods, with extensive research on wider settings, heuristics, and applications [25, 1].

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 25; pp. 25:1–25:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Until recently, all studies of the synthesis problem considered *finite state* transducers that realize specifications given by temporal-logic formulas over a finite set of Boolean propositions or by finite-state automata. Many real-life systems, however, have an infinite state space. One class of infinite-state systems, motivating this work, consists of systems in which the control is finite and the source of infinity is the domain of the variables in the systems. This includes, for example, data-independent programs [37, 20, 27], software with integer parameters [5], communication protocols with message parameters [11], datalog systems with infinite data domain [4, 36], and more [8, 6]. Lifting automata-based methods to the setting of such systems requires the introduction of automata with *infinite alphabets*. The latter include *registers* [33], *pebbles* [28, 34], or *variables* [18, 19], or handle the infinite alphabets by attributing it by labels from an auxiliary finite alphabet [3, 2].

A *register automaton* [33] has a finite set of registers, each of which may contain a letter from the infinite alphabet. The transitions of a register automaton do not refer explicitly to each of the (infinitely many) input letters. Rather, they compare the letter in the input with the content of the registers, and may also store the input letter in a register. Several variants of this model have been studied. For example, [21] forces the content of the registers to be different, [28] adds alternation and two-wayness, [22] allows the registers to change their content nondeterministically during the run, and [35] adds the ability to check for uniqueness of the input letter. Likewise, *register transducers* are adjusted to model systems whose interaction involves input and output variables over an infinite domain: their transitions are labeled by guards that compare the value in the input with the content of the registers. In addition, while taking a transition, the transducer stores this value in some of its registers and outputs a value stored in one of its registers. For example, a transition of a register transducer can be "in state q_5, if the value in the input is not equal to the value stored in register #1, then store the value in the input into register #2, output the value stored in register #1, and transit to state q_3". A register automaton can thus specify properties like "every value read in the input in two successive cycles is output in the next cycle". For more elaborated examples, see Examples 1 and 2.

The transition to infinite alphabets makes reasoning much more complex. In particular, the universality and containment problems for register automata are undecidable [28], and so is the synthesis problem for specifications given by register automata [14]. While the specifications used for the undecidability result in [14] are register automata with a fixed number of registers, the realizing transducers are equipped with an unbounded queue of registers: they can push the inputs into the queue, and later compare the inputs with the values in the queue. This, for example, is helpful for realizing specifications like "every value that appears in the input has to eventually appear on the output twice". While the latter can be specified by a register automaton with a single register, a realizing transducer for it may behave as follows: it queues every incoming value into its queue, outputs the value stored in the head of the queue twice, and dequeues it – which requires an unbounded queue of registers. Moreover, as shown in [15], the synthesis problem stays undecidable even when the number of registers in the realizing transducer is finite, yet not known in advance. In [24], it is shown that bounding the number of registers of the realizing transducer makes the synthesis problem decidable. Essentially, such a bound enables an abstraction of the infinite number of register valuations to a finite number of equivalence relations. In more details, since the transitions of the specification register automaton only compare the value in the input with the content of its registers, we can abstract the exact values stored in the registers and only maintain their partition into equivalence classes: two registers are in the same class if they agree on the values stored in them. In particular, such a partition fixes the transition that the automaton should take, and can be updated whenever the input value is stored in some register.

In this paper we offer a comprehensive study of the synthesis problem for systems with a bounded number of registers. As has been the case with *bounded synthesis* in the finite-state setting [31, 13, 16, 26], the motivation for the study is both conceptual and computational: First, the bounded setting captures better real-life scenarios where bounds on the systems and/or its environment are known. Second, the bounds are the key to new synthesis algorithms, and in the case of systems with an infinite variable domain, they lead to decidability. Note that the only parameter we bound is the number of registers. In particular, the size of the alphabet stays infinite, and the size of the system and its environment stays unbounded[1].

Let us start with the conceptual motivation. It is by now realized that requiring a realizing system to satisfy the specification against all possible environments is often too demanding. Dually, allowing all possible systems is perhaps not demanding enough. This issue is traditionally approached by adding assumptions on the system and/or the environment, which are modeled as part of the specification (see e.g. [9]). In bounded synthesis in the finite-state setting, the assumptions on the system and its environment are given by means of bounds on the sizes of their state space [31, 26]. In the setting of register transducers, bounding the size of the state spaces of the system and its environment is not of much interest, as a register may be used to store the value of the state. Thus, the interesting parameter to bound is the number of allowed registers. Indeed, this setting corresponds to systems with a finite control and a finite number of memory elements, each maintaining a value from an infinite domain. Formally, the *register-bounded realizability problem* is to decide, given a specification register automaton A over infinite input and output alphabets and numbers k_s and k_e of registers, whether there is a system transducer T with at most k_s registers such that for all environment transducers T' with at most k_e registers, the computation $T \| T'$, generated by the interaction of T with T', satisfies the specification A. The *register-bounded synthesis problem* is to construct such a transducer T, if exists.

We continue to the computational motivation and describe our contribution. Our specifications are given by *universal register parity automata on infinite words* (reg-UPW, for short). Thus, each configuration of the automaton may have several successor configurations, and an infinite word is accepted if all the possible runs on it are accepting. Reg-UPWs are more expressive than deterministic register parity automata or universal register Büchi automata, and are more succinct than universal register co-Büchi automata. Reg-UPWs are incomparable with nondeterministic register parity automata (reg-NPW). There are good reasons to work with the universal (rather than nondeterministic) model. First, basic questions are undecidable for reg-NPW. In particular, [12] shows undecidability of the universality problem for nondeterministic register weak automata with a single register, which can be shown to imply undecidability of reg-NPW register-bounded synthesis. Second, as we demonstrate in Section 2, the class of properties that are expressible by reg-UPWs is more interesting in practice. In particular, reg-UPWs are easily closed under conjunction, which is crucial for synthesis.

We describe a simple algorithm for the register-bounded synthesis problem for reg-UPW specifications ([24] only handles co-Büchi automata), which enables a clean complexity analysis ([24] only shows decidability). We study the settings in which both the system and

[1] We note, however, that bounding the number of states in the realizing transducer has proven to be helpful also in the context of systems over infinite alphabets. For example, [17] describes a CEGAR-based synthesis algorithm that approaches the general undecidable synthesis problem by iteratively refining under-approximating systems of bounded sizes.

the environment are bounded ([24] only bounds the system), and we study the theoretical aspects of the setting; in particular, the differences between a fixed, a finite yet unbounded, and an infinite number of registers, and the *determinacy* of the corresponding games.

Our synthesis algorithm reduces the register-bounded synthesis problem to the traditional synthesis problem. Specifically, given a specification reg-UPW A with k_A registers, and numbers k_s and k_e, we construct a (register-less) UPW A' that abstracts the values in the registers of A and consider instead equivalences among registers in the three sets of registers involved: these of A, and these of the system and environment transducers. The synthesis problem for A is then reduced to that of A'. In Section 3 we solve the case where the environment is not bounded (thus $k_e = \infty$) and then in Section 4 continue to the general case. Our complexity analysis carefully takes into account the fact that in the determinization of A', the registers of A and the environment behave universally, whereas these of the system behave deterministically. Accordingly, the complexity of the register-bounded synthesis problem for A with n states, finite alphabet of size m, and index c, can be solved in time $(cmn(k_s + k_e + k_A))^{O(cn(k_s+k_e+k_A)^{(k_e+k_A+1)})}$. Thus, it is polynomial in m, exponential in c, n, and k_s, and doubly-exponential only in k_A and k_e. In the full version [23], we also study *determinacy* of register-bounded synthesis and show that for all $k_s \in \mathbb{N}$ and $k_e \in \mathbb{N} \cup \{\infty\}$, the problem is not determined: there are specifications that are neither realizable by a bounded system (with respect to bounded environments), nor their negations are realizable by a bounded environment (with respect to bounded systems). This corresponds to the picture obtained for bounded synthesis for finite-state systems, where the size of the state space is bounded (we bound only the number of registers) [26]. We also examine the difference in the strength of systems and environments with a fixed, finite, or infinite number of registers, and the existence of a cut-off point, namely a finite-model property characterizing settings where a finite and bounded number of registers suffices.

2 Preliminaries

2.1 Register Automata

Let Σ_I and Σ_O be two finite alphabets and let $\mathcal{D}$ be an infinite domain of *data* values. We consider systems that get inputs in $\Sigma_I \times \mathcal{D}$ and respond with outputs in $\Sigma_O \times \mathcal{D}$. Let $\Sigma = \Sigma_I \times \Sigma_O$. Computations of systems as above are words in $\langle \sigma_0, i_0, o_0 \rangle \langle \sigma_1, i_1, o_1 \rangle ... \in (\Sigma \times \mathcal{D} \times \mathcal{D})^\omega$. *Register automata* specify languages of such words. Let $\mathbb{B} = \{\mathit{true}, \mathit{false}\}$. A *$k$-register word automaton* is a tuple $A = \langle \Sigma, Q, q_0, R, v_0, \delta, \alpha \rangle$, where Σ is a *finite alphabet*, Q is the set of *states*, $q_0 \in Q$ is an *initial* state, R is a set of k *registers*, $v_0 \in \mathcal{D}^R$ is an *initial register valuation*, $\delta : Q \times (\Sigma \times \mathbb{B}^R \times \mathbb{B}^R) \to 2^{Q \times \mathbb{B}^R}$ is a *transition function*, and α is an *acceptance condition* (we later define several acceptance conditions). Intuitively, when A is in state q and reads a letter $\langle \sigma, i, o \rangle \in \Sigma \times \mathcal{D} \times \mathcal{D}$, it compares i and o with the content of its registers and branches into several new configurations according to the result of this comparison. In more detail, rather than specifying a transition for each element in $\Sigma \times \mathcal{D} \times \mathcal{D}$, the transition function δ specifies a transition for each element in $\Sigma \times \mathbb{B}^R \times \mathbb{B}^R$, where the two *guards* in $\mathbb{B}^R$ compare the values stored in the registers with i and o. Then, δ directs A into a set of pairs in $Q \times \mathbb{B}^R$, each describing a successor state and a *storing* mask, indicating which registers are going to store i.

A *configuration of* A is a pair $\langle q, v \rangle \in Q \times \mathcal{D}^R$, describing the state that A visits and the content of its registers. A *run* of A starts in the configuration $\langle q_0, v_0 \rangle$, and continues to form an infinite sequence of successive configurations. In order to define runs formally, we first need some notations. Given a valuation $v \in \mathcal{D}^R$ and a value $d \in \mathcal{D}$, let $v \sim d$ denote the

Boolean assignment $g \in \mathbb{B}^R$ that indicates the agreement of $\mathpzc{v}$ with $\mathpzc{d}$. Thus, for every $r \in R$, we have $g(r) = \mathit{true}$ iff $\mathpzc{v}(r) = \mathpzc{d}$. The function $\mathit{update} : \mathcal{D}^R \times \mathcal{D} \times \mathbb{B}^R \to \mathcal{D}^R$ maps a valuation $\mathpzc{v} \in \mathcal{D}^R$, a value $\mathpzc{d} \in \mathcal{D}$, and a *storing mask* $a \in \mathbb{B}^R$, to the valuation obtained from $\mathpzc{v}$ by changing the value stored in registers that are positive in a to $\mathpzc{d}$. Formally, for every $r \in R$, we have that $\mathit{update}(\mathpzc{v}, \mathpzc{d}, a)(r)$ is d if $a(r) = \mathit{true}$ and is $\mathpzc{v}(r)$ otherwise. Note that it need not be the case that $\mathit{update}(\mathpzc{v}, \mathpzc{d}, a) \sim \mathpzc{d} = a$. Indeed, if $\mathpzc{v}(r) = \mathpzc{d}$, then $\mathit{update}(\mathpzc{v}, \mathpzc{d}, a)(r) = \mathpzc{d}$ regardless of $a(r)$.

For two configurations $\langle q', \mathpzc{v}' \rangle$ and $\langle q, \mathpzc{v} \rangle$ in $Q \times \mathcal{D}^R$, and a triple $\langle \sigma, i, o \rangle \in \Sigma \times \mathcal{D} \times \mathcal{D}$, we say that $\langle q', \mathpzc{v}' \rangle$ is a *$\langle \sigma, i, o \rangle$-successor of* $\langle q, \mathpzc{v} \rangle$ if there exists $a \in \mathbb{B}^R$ such that $\langle q', a \rangle \in \delta(q, \langle \sigma, \mathpzc{v} \sim i, \mathpzc{v} \sim o \rangle)$ and $\mathpzc{v}' = \mathit{update}(\mathpzc{v}, i, a)$.

Now, a *run* of A on a word $w = \langle \sigma_0, i_0, o_0 \rangle \langle \sigma_1, i_1, o_1 \rangle ... \in (\Sigma \times \mathcal{D} \times \mathcal{D})^\omega$ is an infinite sequence $\langle q_0, \mathpzc{v}_0 \rangle \langle q_1, \mathpzc{v}_1 \rangle ... \in (Q \times \mathcal{D}^R)^\omega$ of configurations such that for every $j \geq 0$, we have that $\langle q_{j+1}, \mathpzc{v}_{j+1} \rangle$ is a $\langle \sigma_j, i_j, o_j \rangle$-successor of $\langle q_j, \mathpzc{v}_j \rangle$. Note that there may be several different runs on the same word. Note also that since δ may return an empty set of possible transitions, a configuration $\langle q_j, \mathpzc{v}_j \rangle$ need not have $\langle \sigma_j, i_j, o_j \rangle$-successors. There, the sequence of successive configurations is finite, and is not a run.

When A is a *parity* automaton, $\alpha : Q \to \{0, ..., c-1\}$, for an *index* $c \in \mathbb{N}$, a run ρ is *accepting* if the maximal rank that is visited by ρ infinitely often is even. Formally, $\rho = \langle q_0, \mathpzc{v}_0 \rangle \langle q_1, \mathpzc{v}_1 \rangle ...$ is *accepting* if $\max\{j \in \{0, ..., c-1\} : \alpha(q_l) = j \text{ for infinitely many } l \geq 0\}$ is even. The *co-Büchi* acceptance condition is a special case of parity, with $c = 2$. Thus, ρ is accepting if vertices $\langle q, \mathpzc{v} \rangle$ with $\alpha(q) = 1$ are visited only finitely often. When A is *universal*, it accepts the word w if all the runs of A on w are accepting. Note that since we require runs to be infinite, the universal quantification on the runs means that a configuration with no successors is like an accepting configuration: once we reach it, there are no restrictions on the suffix of the word. The language of A, denoted $L(A)$, is the set of all words that A accepts. We sometimes use $w \models A$ to indicate that $w \in L(A)$. We use *reg-UPW* and *reg-UCW* to abbreviate a universal register parity and co-Büchi automata, respectively. A (register-less) *UPW* can be viewed as a special case of a reg-UPW with no registers. In particular, it has no initial valuation and its transition function is of the form $\delta : Q \times \Sigma \to 2^Q$.

▶ **Example 1.** The reg-UCW A appearing in Figure 1 specifies an arbiter with a single output signal *ack* (that is, Σ_I is a singleton, and we ignore it, and $\Sigma_O = 2^{\{ack\}}$) that gets in each moment in time an input data value i, and outputs either *ack* or $\neg ack$ along with an output data value o. It accepts a word if every input data value different from the previous one is eventually outputted with *ack*. The acceptance condition α requires runs to visit q_1 only finitely often. The reg-UCW A has a single register, thus $R = \{r_1\}$, and we describe vectors in $\Sigma \times \mathbb{B}^R \times \mathbb{B}^R$ by triples in $\{ack, \neg ack\} \times \{0, 1\} \times \{0, 1\}$, possibly replacing some of the parameters by _, indicating that both values of this parameter apply. We continue to describe

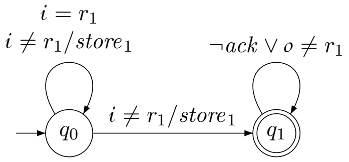


Figure 1 The reg-UCW A. The edge labels are symbolic, where the expressions $i \neq r_1$ and $i = r_1$ mean that the i-guard is 0 and 1 respectively, and the expression $o \neq r_1$ means that the o-guard is 0. The label $store_1$ means the storing mask is 1, while its absence means it is 0. The state q_1 is doubly-circled, indicating that a run is accepting iff it visits q_1 only finitely often.

the transition function. First, $\delta(q_0, \langle \_, 1, \_\rangle) = \{\langle q_0, 0\rangle\}$. That is, if the input data value agrees with the one stored in r_1, we only loop in q_0. Then, $\delta(q_0, \langle \_, 0, \_\rangle) = \{\langle q_0, 1\rangle, \langle q_1, 1\rangle\}$. That is, if the input data value differs from the one stored in r_1, then A both loops in q_0 and sends a copy to q_1, and stores the value of the input data value in r_1. In state q_1, we have $\delta(q_1, \langle ack, \_, 1\rangle) = \varnothing$, thus the copy sent to q_1 fulfils its mission when it reads an ack with an output data value that agrees with the one stored in r_1. In all other cases, the copy stays in q_1. Thus, $\delta(q_1, \langle \neg ack, \_, \_\rangle) = \delta(q_1, \langle ack, \_, 0\rangle) = \langle q_1, 0\rangle$. The parity acceptance condition $\alpha = \{q_0 \mapsto 0, q_1 \mapsto 1\}$ then guarantees that all copies sent to q_1 eventually fulfil their missions. We note that the universality of A is used in order to detect all data values that are not stored in r_1: a copy of the automaton is launched for each of them. Such a detection is impossible in a deterministic or even a nondeterministic register automaton.

2.2 Register Transducers

Register transducers model systems with inputs in $\Sigma_I \times \mathcal{D}$ and outputs in $\Sigma_O \times \mathcal{D}$. Every such system implements a *strategy* $(\Sigma_I \times \mathcal{D})^+ \to \Sigma_O \times \mathcal{D}$, describing the output it generates after reading a sequence of inputs. A *register transducer* is a tuple $T = \langle \Sigma_I, \Sigma_O, S, s_0, R, v_0, \tau\rangle$, where Σ_I and Σ_O are *input* and *output finite alphabets*, S is a set of *states*, $s_0 \in S$ is an *initial state*, R is a set of *registers*, $v_0 \in \mathcal{D}^R$ is an *initial register valuation*, and $\tau : S \times (\Sigma_I \times \mathbb{B}^R) \to S \times \mathbb{B}^R \times \Sigma_O \times R$ is a *transition function*. Intuitively, when T is in state s and reads a letter $\langle i, \imath\rangle \in \Sigma_I \times \mathcal{D}$, it compares $\imath$ with the content of its registers. Depending on i and the comparison, it transits deterministically to a successor state and may store the data value $\imath$ into its registers. It also outputs a letter in Σ_O and a value stored in one of the registers. Note that a register may store either its initial value or some value seen earlier as a data input.

Formally, a configuration of T is a pair in $S \times \mathcal{D}^R$, and successive configurations are defined in a way similar to the one defined for automata, except that T is deterministic: given a configuration $\langle s, v\rangle \in S \times \mathcal{D}^R$ and an input $\langle i, \imath\rangle \in \Sigma_I \times \mathcal{D}$, let $\tau(s, \langle i, v \sim \imath\rangle) = \langle s', a, o, r\rangle$. Then, the $\langle i, \imath\rangle$-*successor* of $\langle s, v\rangle$ is $\langle s', \mathit{update}(v, \imath, a)\rangle$.

Given an input word $w = \langle i_0, \imath_0\rangle\langle i_1, \imath_1\rangle... \in (\Sigma_I \times \mathcal{D})^\omega$, the *run* of T on w is the sequence $\langle s_0, v_0\rangle\langle s_1, v_1\rangle... \in (S \times \mathcal{D}^R)^\omega$, where for all $j \geq 0$, we have that $\langle s_{j+1}, v_{j+1}\rangle$ is the $\langle i_j, \imath_j\rangle$-successor of $\langle s_j, v_j\rangle$. For every $j \geq 0$, let $\tau(s_j, i_j, v_j \sim \imath_j) = \langle s_{j+1}, a_j, o_j, r_j\rangle$. Then, the *computation* of T on w is the sequence $\langle\langle i_0, o_0\rangle, \imath_0, o_0\rangle\langle\langle i_1, o_1\rangle, \imath_1, o_1\rangle... \in ((\Sigma_I \times \Sigma_O) \times \mathcal{D} \times \mathcal{D})^\omega$ such that for every $j \geq 0$, we have that $o_j = \mathit{update}(v_j, \imath_j, a_j)(r_j)$. Thus, the transducer moves from s_j to s_{j+1}, stores $\imath_j$ in registers that are positive in a_j, and then outputs o_j and the (updated) content of register r_j. A (register-less) transducer is a special case of a register transducer with no registers. In particular, it has no initial valuation and its transition function is of the form $\tau : S \times \Sigma_I \to S \times \Sigma_O$.

For a register transducer T and a reg-UPW A, we say that T *realizes* A, denoted $T \models A$, if for all input words $w \in (\Sigma_I \times \mathcal{D})^\omega$, the computation of T on w is in the language of A.

▶ **Example 2.** Figure 2 describes a register transducer that realizes the reg-UCW from Example 1. The input alphabet Σ_I is a singleton and we ignore it. The output alphabet $\Sigma_O = 2^{\{ack\}}$, and the register set $R = \{r_1, r_2\}$. The transducer loops in the initial state s_0 if the current data input equals the previous data input (which is stored in register r_1). Otherwise ($\imath \neq r_1$), the transducer stores the new data value into r_1, does not raise *ack*, outputs the value of register r_1 (it has to output something), and moves into state s_1. Now, if it does not see a new data input ($\imath = r_1$), then – in order to acknowledge the previous data input – it raises *ack*, outputs the previous data input from r_1, and returns into s_0.

Alternatively, if in state s_1 the transducer sees a new data input ($i \neq r_1$), then it stores into r_2, raises *ack*, outputs the previous data input from r_1, and moves into s_2. From there, if no new data input was seen, the transducer moves into s_3, while outputting the value of r_2 and raising *ack*. And so on. Thus, in states s_0 and s_1 register r_1 contains the previous data input, while in states s_2 and s_3 it is stored in register r_2. Finally, register r_1 is initialized with the same value as the automaton register, while r_2 can start with anything. We conclude with a remark that there is a simpler transducer that realizes the same reg-UCW: It always raises *ack*, stores alternatingly into r_1 and r_2 while outputting alternatingly the value of r_2 and r_1. But such a transducer produces spurious *ack*s, while our transducer does not.

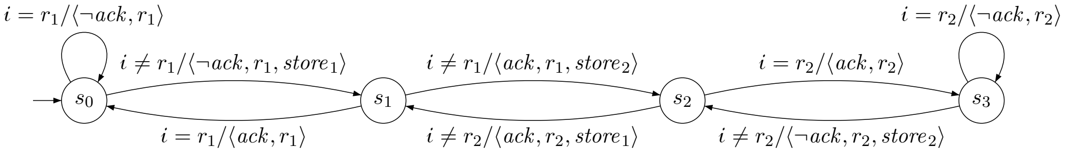

$i = r_1/\langle\neg ack, r_1\rangle$
$i = r_2/\langle\neg ack, r_2\rangle$
$i \neq r_1/\langle\neg ack, r_1, store_1\rangle$
$i \neq r_1/\langle ack, r_1, store_2\rangle$
$i = r_2/\langle ack, r_2\rangle$
s_0
s_1
s_2
s_3
$i = r_1/\langle ack, r_1\rangle$
$i \neq r_2/\langle ack, r_2, store_1\rangle$
$i \neq r_2/\langle\neg ack, r_2, store_2\rangle$

Figure 2 A register transducer that realizes the reg-UCW A from Example 1. The edge labeling for Σ_O and the guards is symbolic, and is similar to that in Figure 1.

2.3 Synthesis with an Infinite or Unbounded Number of System Registers

The *realizability problem* is to decide, given a reg-UPW A over $\Sigma_I \times \Sigma_O \times \mathcal{D} \times \mathcal{D}$, whether there is a register transducer all whose computations are accepted by A. The *synthesis problem* is to construct such a transducer, if exists.

The realizability and synthesis problems in the context of specifications and systems with an infinite data domain was first studied in [14]. The transducers in [14] have an infinite number of registers, all initialized to the same value. The automata in [14] are universal register automata with a variant of weak acceptance condition, and additionally do not allow for register re-assignment. It is shown in [14] that the synthesis problems is undecidable, already for automata with only two registers. Since our automata and transducers are more powerful, undecidability applies to our setting. Thus, when the number of registers in the system is infinite, the realizability and synthesis problems are undecidable.

Consider now the case where the number of registers is finite but not fixed a-priori. It is shown in [12] that the nonemptiness problem for universal 2-register automata on finite words is undecidable. It is not hard to reduce their nonemptiness problem to the synthesis problem for 2-register UPWs, which implies the undecidability of the latter. Thus, we get the following.

▶ **Theorem 3** ([12, 14]). *The synthesis problem of transducers with an infinite or a finite but unbounded number of registers for specifications given by* 2*-register UPWs is undecidable. In the case of an infinite number of registers, undecidability holds even when the transducer registers are initialized with the same value.*

3 Synthesis with a Fixed Number of System Registers

The *system-bounded realizability problem* is to decide, given a reg-UPW A over $\Sigma_I \times \Sigma_O \times \mathcal{D} \times \mathcal{D}$ and a number k_s of registers, whether there is a transducer with at most k_s registers all whose computations are accepted by A. The *system-bounded synthesis problem* is to construct such a transducer, if exists.

Let $A = \langle \Sigma, Q, q_0, R_A, \mathfrak{v}_0^A, \delta, \alpha \rangle$, and let $|R_A| = k_A$. Recall that $\Sigma = \Sigma_I \times \Sigma_O$. We define a UPW A' (that is, with no registers) that abstracts the values stored in R_A. Instead, A' maintains an equivalence relation over the registers of A and the registers of the realizing transducer, indicating which of them agree on the values stored in them.

Let R_s denote a set of k_s registers, namely these of the realizing transducer (we subscript its elements by s as this transducer models the system), and let $R = R_A \cup R_s$. For valuations $\mathfrak{v}_A \in \mathcal{D}^{R_A}$ and $\mathfrak{v}_s \in \mathcal{D}^{R_s}$, let $\mathfrak{v}_A \cup \mathfrak{v}_s$ be the valuation in $\mathcal{D}^R$ obtained by taking their union. Likewise, for a valuation $\mathfrak{v} \in \mathcal{D}^R$, let $\mathfrak{v}_A$ and $\mathfrak{v}_s$ denote the projections of $\mathfrak{v}$ on R_A and R_s, respectively. Let Π be the set of all equivalence relations over R. Consider an element $\pi \in \Pi$, thus $\pi \subseteq R \times R$. For two registers $r, r' \in R$, we write $\pi(r, r')$ to denote that r and r' are equivalent in π. Note that r and r' may be both in R_A, both in R_s, or one in R_A and one in R_s. Each equivalence relation $\pi \in \Pi$ induces a partition of R into equivalence classes, and we sometimes refer to the elements in Π as partitions of R. Then, for $\pi \in \Pi$, we talk about sets $S \in \pi$, where $S \subseteq R$, and $\pi(r, r')$ indicates that r and r' are in the same set in the partition. Let $f : \mathcal{D}^R \to \Pi$ map a register valuation $\mathfrak{v} \in \mathcal{D}^R$ to the partition $\pi \in \Pi$, where for every two registers $r, r' \in R$, we have that $\pi(r, r')$ iff $\mathfrak{v}(r) = \mathfrak{v}(r')$.

Recall that we describe guards and storing masks on a set R of registers by Boolean functions in $\mathbb{B}^R$. Each assignment $g \in \mathbb{B}^R$ corresponds to a set of registers characterized by g. In the sequel, we sometimes refer to Boolean assignments as sets, thus assume that $g \subseteq R$, and talk about union and intersection of assignments, referring to the sets they characterize. Consider a partition π of R and a Boolean assignment $g_s \subseteq R_s$. We say that g_s is *π-consistent* if there is an equivalence class $S \in \pi \cup \{\varnothing\}$ such that $S \cap R_s = g_s$. We then say that $\langle \pi, g_s \rangle$ *chooses* S. Note that for $g_s = \varnothing$, the set S is either empty or contains no system registers, and might be not unique. For example, if $R_A = \{\#1, \#2, \#3, \#4\}$, $R_s = \{\#5, \#6\}$, and $\pi = \{\{\#1\}, \{\#2, \#3\}, \{\#4, \#5\}, \{\#6\}\}$, then $\langle \pi, \{\#5\} \rangle$ chooses only $\{\#4, \#5\}$, the pair $\langle \pi, \{\#6\} \rangle$ chooses only $\{\#6\}$, and $\langle \pi, \varnothing \rangle$ chooses $\{\#1\}$, $\{\#2, \#3\}$, or $\varnothing$. For a set $S_A \subseteq R_A$, we say that $\langle \pi, g_s \rangle$ *A-chooses* S_A if there is a set $S \in \pi \cup \{\varnothing\}$ such that $\langle \pi, g_s \rangle$ chooses S and $S_A = S \cap R_A$. Thus, $\langle \pi, g_s \rangle$ A-chooses S_A if $\langle \pi, g_s \rangle$ chooses a set whose R_A registers are these in S_A. Continuing the previous example, $\langle \pi, \{\#5\} \rangle$ A-chooses $\{\#4\}$, the pair $\langle \pi, \{\#6\} \rangle$ A-chooses $\varnothing$, and $\langle \pi, \varnothing \rangle$ A-chooses $\{\#1\}$, $\{\#2, \#3\}$, or $\varnothing$. Finally, for a register $r \in R$, the pair $\langle \pi, r \rangle$ *A-chooses* the unique set $S_A \subseteq R_A$ if $S_A = S \cap R_A$, for the set $S \in \pi$ such that $r \in S$. In the example above, the pairs $\langle \pi, \#4 \rangle$ and $\langle \pi, \#5 \rangle$ both A-choose $\{\#4\}$, and the pair $\langle \pi, \#6 \rangle$ A-chooses $\varnothing$.

The following lemma follows immediately from the definitions.

▶ **Lemma 4.** *Consider a partition π of $R = R_s \cup R_A$ and a valuation $\mathfrak{v} \in \mathcal{D}^R$ s.t. $f(\mathfrak{v}) = \pi$. Then:*

(a) *for every $i \in \mathcal{D}$, the guard $\mathfrak{v}_s \sim i$ is π-consistent and A-chooses the guard $\mathfrak{v}_A \sim i$,*

(b) *for every guard $g \in (\pi \cup \{\varnothing\})$, there exists $i \in \mathcal{D}$ satisfying $(\mathfrak{v} \sim i) = g$, and*

(c) *for every $r \in R$, the pair $\langle \pi, r \rangle$ A-chooses $\mathfrak{v}_A \sim \mathfrak{v}(r)$.*

Recall the function $\mathit{update} : \mathcal{D}^R \times \mathcal{D} \times \mathbb{B}^R \to \mathcal{D}^R$, where $\mathit{update}(\mathfrak{v}, \mathscr{d}, a)$ is obtained from $\mathfrak{v}$ by storing $\mathscr{d}$ in the registers in a. We now define a function $\mathit{update}' : \Pi \times \mathbb{B}^R \times \mathbb{B}^R \to \Pi$, which adjusts the update function to the abstraction of valuations by partitions. Intuitively, for a partition $\pi \in \Pi$, a guard $g \in (\pi \cup \{\varnothing\})$, and a storing mask $a \subseteq R$, we obtain the partition $\mathit{update}'(\pi, g, a)$ from π by moving the registers in a either into the equivalence class of g (if g is not empty), or into a new equivalence class. Formally, $\mathit{update}'(\pi, g, a) = \{S \setminus a : S \in \pi \setminus g\} \setminus \{\varnothing\} \cup \{g \cup a\}$. Note that, in particular, $\mathit{update}'(\pi, \varnothing, a) = \{S \setminus a : S \in \pi\} \setminus \{\varnothing\} \cup \{a\}$.

▶ **Lemma 5.** *For every valuation $\mathfrak{v} \in \mathcal{D}^R$, value $i \in \mathcal{D}$, and storing mask $a \subseteq R$, we have that $f(update(\mathfrak{v}, i, a)) = update'(f(\mathfrak{v}), \mathfrak{v} \sim i, a)$.*

We are now ready to define the abstraction of A. In addition to k_s, the abstraction is parameterized by a partition π_0 of the system and automaton registers. Given k_s and π_0, the (k_s, π_0)-*abstraction of* A is the UPW $A' = \langle \Sigma', Q', q'_0, \delta', \alpha' \rangle$ with the following components.

- $Q' = Q \times \Pi$ and $q'_0 = \langle q_0, \pi_0 \rangle$. Thus, each state in A' is a pair $\langle q, \pi \rangle$, abstracting configurations $\langle q, \mathfrak{v}_A \rangle$ of A and register valuations $\mathfrak{v}_s$ of an anticipated transducer that satisfy $f(\mathfrak{v}_s \cup \mathfrak{v}_A) = \pi$.
- $\Sigma' = \Sigma \times \mathbb{B}^{R_s} \times R_s \times \mathbb{B}^{R_s}$. Recall that in A, the transition function is $\delta : Q \times (\Sigma \times \mathbb{B}^{R_A} \times \mathbb{B}^{R_A}) \to 2^{Q \times \mathbb{B}^{R_A}}$, and when A is in configuration $\langle q, \mathfrak{v}_A \rangle$ and reads a letter $\langle \sigma, i, o \rangle \in \Sigma \times \mathcal{D} \times \mathcal{D}$, it proceeds according to $\langle \sigma, g_i^A, g_o^A \rangle \in \Sigma \times \mathbb{B}^{R_A} \times \mathbb{B}^{R_A}$, where g_i^A is $\mathfrak{v}_A \sim i$ and g_o^A is $\mathfrak{v}_A \sim o$. Also, each successor state q' is paired with a storing mask $a_i^A \in \mathbb{B}^{R_A}$, which induces a successor configuration $\langle q', update(\mathfrak{v}, i, a_i^A) \rangle$. Intuitively, each letter $\langle \sigma, g_i^s, r_s, a_i^s \rangle \in \Sigma'$, together with the current partition, induces choices for $\langle \sigma, g_i^A, g_o^A, a_i^A \rangle \in \Sigma \times \mathbb{B}^{R_A} \times \mathbb{B}^{R_A} \times \mathbb{B}^{R_A}$ which determine the transitions in A that the abstraction follows.
- For every state $\langle q, \pi \rangle \in Q'$ and letter $\langle \sigma, g_i^s, r_s, a_i^s \rangle \in \Sigma'$, we have that $\langle q', \pi' \rangle \in \delta'(\langle q, \pi \rangle, \langle \sigma, g_i^s, r_s, a_i^s \rangle)$ iff there exist $g_i^A, g_o^A, a_i^A \in \mathbb{B}^{R_A}$ such that the following conditions hold.
  - g_i^A is A-chosen by $\langle \pi, g_i^s \rangle$. Let $g_i = g_i^s \cup g_i^A$. Note that $g_i \in (\pi \cup \{\varnothing\})$.
  - Recall that the output value in register transducers refers to the updated register values, namely their values in the successor configuration. Therefore, when we compare the data output of a transducer with the register values of the automaton, we first have to update the values of the system transducer. For this, we introduce the partition $\pi^\star$. Let $\pi^\star$ be the partition after updating the system registers in π according to the guard g_i^s and the storing mask a_i^s. Thus, $\pi^\star = update'(\pi, g_i, a_i^s)$.
  - g_o^A is A-chosen by $\langle \pi^\star, r_s \rangle$. Note that since the set chosen by $\langle \pi^\star, r_s \rangle$ is not empty, g_o^A is unique.
  - $\langle q', a_i^A \rangle \in \delta(q, \langle \sigma, g_i^A, g_o^A \rangle)$.
  - We can now complete updating the partition. The partition π' is the result of updating the registers of A in $\pi^\star$ according to the guard g_i^A and the storing mask a_i^A. Let $g_i^\star = g_i \cup a_i^s$ be the updated guard after system storing. Then $\pi' = update'(\pi^\star, g_i^\star, a_i^A)$.
- The acceptance condition of A' is induced from the one of A. Thus, for every state $\langle q, \pi \rangle \in Q'$, we have that $\alpha'(\langle q, \pi \rangle) = \alpha(q)$.

Recall that the abstraction of A is parameterized by both the number of registers that the system transducer may have as well as an initial partition for the registers of both the system and the automaton. Let $\mathfrak{v}_A \in \mathcal{D}^{R_A}$ be a valuation of the automaton registers. A partition $\pi \in \Pi$ is *consistent with* $\mathfrak{v}_A$ if there is a register valuation $\mathfrak{v}_s \in \mathcal{D}^{R_s}$ such that $\pi = f(\mathfrak{v}_A \cup \mathfrak{v}_s)$. Thus, all automaton registers are related according to $\mathfrak{v}_A$, and the system registers are unrestricted.

▶ **Example 6.** Let $\mathcal{D} = \mathbb{N}$, $R_A = \{\#1, \#2, \#3, \#4\}$, and $R_s = \{\#5, \#6, \#7\}$. Then the partition $\pi = \{\{\#1, \#4, \#5\}, \{\#2, \#6\}, \{\#3\}, \{\#7\}\}$ is consistent with the valuation $\mathfrak{v}_A \in \mathcal{D}^{R_A}$ for which $\mathfrak{v}_A(\#1) = \mathfrak{v}_A(\#4) = 9$, $\mathfrak{v}_A(\#2) = 2$, and $\mathfrak{v}_A(\#3) = 13$. Indeed, taking $\mathfrak{v}_s \in \mathcal{D}^{R_s}$ with $\mathfrak{v}_s(\#5) = 9$, $\mathfrak{v}_s(\#6) = 2$, and $\mathfrak{v}_s(\#7) = 14$ results in $\pi = f(\mathfrak{v}_A \cup \mathfrak{v}_s)$. Note that different valuations $\mathfrak{v}_s \in \mathcal{D}^{R_s}$ may witness the consistency of π with $\mathfrak{v}_A$. In our example, all these with $\mathfrak{v}_s(\#5) = 9$, $\mathfrak{v}_s(\#6) = 2$, and $\mathfrak{v}_s(\#7) \notin \{2, 9, 13\}$. Also, several different partitions may be consistent with a given valuation $\mathfrak{v}_A \in \mathcal{D}^{R_A}$. In our example, all these in which register #1 and #4 are in the same set, different from the (different) sets of #2 and #3.

We can now state our main theorem, relating the realizability of A with realizability of its abstraction. Consider a k_s-register Σ_I/Σ_O-transducer $T = \langle \Sigma_I, \Sigma_O, S, s_0, R, \mathpzc{v}_0, \tau\rangle$. We can view T as a (register-less) Σ'_I/Σ'_O–transducer T', for $\Sigma'_I = \Sigma_I \times \mathbb{B}^R$ and $\Sigma'_O = \mathbb{B}^R \times \Sigma_O \times R$. Indeed, the transition function $\tau : S \times (\Sigma_I \times \mathbb{B}^R) \to S \times \mathbb{B}^R \times \Sigma_O \times R$ of T can be viewed as $\tau' : S \times \Sigma'_I \to S \times \Sigma'_O$. When $\mathpzc{v}_0 \in \mathscr{D}^{R_s}$ is fixed, we say that T and T' *correspond* to each other. Essentially, our main theorem follows from the fact that a reg-UPW A is realized by a k_s-transducer T iff the abstraction of A is realized by the register-less transducer that corresponds to T. Formally, we have the following.

▶ **Theorem 7.** *Consider a reg-UPW A with $\Sigma = \Sigma_I \times \Sigma_O$, set of registers R_A, and an initial valuation $\mathpzc{v}^A_0$. Then, A is realizable by a k_s-register Σ_I/Σ_O-transducer with a set of registers R_s iff there is a partition π_0 of $R = R_s \cup R_A$, consistent with $\mathpzc{v}^A_0$, such that the (k_s, π_0)-abstraction of A is realizable by a $(\Sigma_I \times \mathbb{B}^{R_s})/(\Sigma_O \times R_s \times \mathbb{B}^{R_s})$-transducer. In particular, a transducer that realizes the (k_s, π_0)-abstraction of A corresponds to a k_s-register transducer that realizes A.*

Proof sketch. Let $A = \langle \Sigma, Q, q_0, R_A, \mathpzc{v}^A_0, \delta, \alpha\rangle$ and let A' be its (k_s, π_0)-abstraction, where π_0 is a partition of R consistent with $\mathpzc{v}^A_0$. We prove that for every valuation $\mathpzc{v}^s_0 \in \mathscr{D}^{R_s}$ satisfying $f(\mathpzc{v}^A_0 \cup \mathpzc{v}^s_0) = \pi_0$, k_s-register Σ_I/Σ_O-transducer T initialized with $\mathpzc{v}^s_0$, and register-less $(\Sigma_I \times \mathbb{B}^{R_s})/(\mathbb{B}^{R_s} \times \Sigma_O \times R_s)$-transducer T', where T and T' correspond to each other, it holds that $T \models A$ iff $T' \models A'$. The theorem then follows.

Assume first that $T \not\models A$. We prove that $T' \not\models A'$. Since $T \not\models A$, there is an input sequence $w^I_T = \langle i_0, \mathpzc{i}_0\rangle\langle i_1, \mathpzc{i}_1\rangle...$, a run $\rho_T = \langle s_0, \mathpzc{v}^s_0\rangle\langle s_1, \mathpzc{v}^s_1\rangle...$ of T on w^I_T, a computation $w_T = \langle\langle i_0, o_0\rangle, \mathpzc{i}_0, \mathpzc{o}_0\rangle\langle\langle i_1, o_1\rangle, \mathpzc{i}_1, \mathpzc{o}_1\rangle...$ that T generates when it follows ρ_T, and a rejecting run $\rho_A = \langle q_0, \mathpzc{v}^A_0\rangle\langle q_1, \mathpzc{v}^A_1\rangle...$ of A on the computation w_T. Note that A may have several runs on w_T. Since it is universal, and A rejects w_T, we know that at least one of them does not satisfy α. We show that w^I_T and ρ_T induce an input sequence $w^I_{T'}$ to T' such that A' rejects the computation of T' on $w^I_{T'}$. We define $w^I_{T'} = \langle i_0, \mathpzc{v}^s_0 \sim \mathpzc{i}_0\rangle\langle i_1, \mathpzc{v}^s_1 \sim \mathpzc{i}_1\rangle....$ The word $w^I_{T'}$ uniquely defines the computation $w_{T'}$ and the run $\rho_{T'} = s_0 s_1...$ of T'. We now define the rejecting run $\rho_{A'}$ of A' on $w_{T'}$. It starts in the configuration $\langle q_0, \pi_0\rangle$. Suppose that in step $j \geq 0$, the run ρ_A reaches the configuration $\langle q, \mathpzc{v}_A\rangle$, the run ρ_T reaches the configuration $\langle s, \mathpzc{v}_s\rangle$, and the run $\rho_{A'}$ reaches the state $\langle q, \pi\rangle$. Assume that $\pi = f(\mathpzc{v}_A \cup \mathpzc{v}_s)$. Since $\pi_0 = f(\mathpzc{v}^A_0 \cup \mathpzc{v}^s_0)$, this holds for $j = 0$. Assume that in ρ_T, the transducer T transit in the step j from $\langle s, \mathpzc{v}_s\rangle$ to $\langle s', \mathpzc{v}'_s\rangle$, while reading $\langle i, \mathpzc{i}\rangle$ and outputting $\langle o, \mathpzc{o}\rangle$. Note that the respective letter of the computation $w_{T'}$ is $\sigma' = \langle\langle i, o\rangle, g^s_\mathpzc{i}, r_s, a_s\rangle$, where $g^s_\mathpzc{i} = (\mathpzc{v}_s \sim \mathpzc{i})$ and it holds that $\langle s', a_s, o, r_s\rangle = \tau(s, i, g^s_\mathpzc{i})$. Let $\langle q', \mathpzc{v}'_A\rangle$ be a $\langle\langle i, o\rangle, \mathpzc{i}, \mathpzc{o}\rangle$-successor of $\langle q, \mathpzc{v}_A\rangle$ as appears in ρ_A. In the full version [23], we prove that the pair $\langle q', \pi'\rangle$ is a σ'-successor of $\langle q, \pi\rangle$ in A', where $\pi' = f(\mathpzc{v}'_A \cup \mathpzc{v}'_s)$. By repeatedly applying the above claim, we can start from $\langle q_0, \pi_0\rangle$ and, for all $j \geq 0$, get the successor $\langle q_{j+1}, \pi_{j+1}\rangle$ of $\langle q_j, \pi_j\rangle$, obtaining the sought run $\rho_{A'}$. Also, by the definition of α', the fact ρ_A is rejecting implies that so is $\rho_{A'}$, and so we are done.

Assume now that $T' \not\models A'$. We prove that $T \not\models A$. Since $T' \not\models A'$, there is an input sequence $w^I_{T'}$ that induces the run $\rho_{T'} = s_0 s_1...$ and the computation $w_{T'}$ of T' such that $w_{T'}$ generates a rejecting run $\rho_{A'} = \langle q_0, \pi_0\rangle\langle q_1, \pi_1\rangle...$ in A'. Given $w_{T'}$ (and hence $\rho_{T'}$) and $\rho_{A'}$, we construct a computation w_T of T that induces a rejecting run ρ_A in A. The run ρ_T starts in $\langle s_0, \mathpzc{v}^s_0\rangle$, and the run ρ_A starts in $\langle q_0, \mathpzc{v}^A_0\rangle$. Suppose that in some step $j \geq 0$, the run $\rho_{T'}$ reaches a state s, the run $\rho_{A'}$ reaches a state $\langle q, \pi\rangle$, the run ρ_T reaches a configuration $\langle s, \mathpzc{v}_s\rangle$, and the run ρ_A reaches a configuration $\langle q, \mathpzc{v}_A\rangle$. Assume that $\pi = f(\mathpzc{v}_s \cup \mathpzc{v}_A)$. This holds for $j = 0$. Assume that T' transits into s' when reading $\langle i, g^s_\mathpzc{i}\rangle$ and outputting $\langle a_s, o, r_s\rangle$, and that A' transits into $\langle q', \pi'\rangle$ when reading $\langle\langle i, o\rangle, g^s_\mathpzc{i}, r_s, a_s\rangle$. Then, as we prove in the full version [23], there exist $i \in \mathcal{D}$ such that the transducer T transits into $\langle s', \mathfrak{v}'_s \rangle$ on reading $\langle i, i \rangle$, the automaton A transits into $\langle q', \mathfrak{v}'_A \rangle$ on reading $\langle \langle i, o \rangle, i, o \rangle$, where $o = \mathfrak{v}'_s(r_s)$, and $f(\mathfrak{v}'_s \cup \mathfrak{v}'_A) = \pi'$. Applying the above claim in the initial step, when $j = 0$, we construct the configuration $\langle s_1, \mathfrak{v}^s_1 \rangle$ of ρ_T, the configuration $\langle q_1, \mathfrak{v}^A_1 \rangle$ of ρ_A, and the first letter $\langle \langle i, o \rangle, i, o \rangle$ of w_T. Note that the claim preconditions hold, in particular, $f(\mathfrak{v}^s_1 \cup \mathfrak{v}^A_1) = \pi_1$, so we can apply it again. By an iterative application, we construct the sought computation w_T and the rejecting run ρ_A on w_T. ◀

We can now analyze the complexity of our synthesis algorithm. Recall that the input to the problem is a reg-UPW A and an integer $k_s \geq 0$, and the output is a k_s-register transducer that realizes A, or an answer that no such transducer exists. Theorem 7 reduces the problem for A with n states, index c, and k_A registers, to the synthesis problem of a (register-less) UPW A' with $n(k_A + k_s)^{k_A + k_s}$ states and index c. Indeed, the state space of A' is the product of that of A with the set of possible partitions of the registers of A and these of the generated transducer, and the number of such partitions is bounded by $(k_A + k_s)^{k_A + k_s}$. Note that A' is parameterized by both k_s and π_0. While k_s is fixed, π_0 depends on the initial partition of R_s. Thus, we may need to repeat the reduction $|\Pi_s| \leq k_s^{k_s}$ times, where Π_s is the set of system partitions. By [29, 32] a UPW with N states and index c can be determinized to a DPW with $(Nc)^{O(Nc)}$ states and index $O(Nc)$. Then, the synthesis problem for DPW reduces linearly, up to a multiplicative factor in the sizes of the alphabets, to solving parity games, which can be done in time at most $O((n')^5)$, for a game with n' vertices and index $c' < \log n'$ [7]. The alphabet of A' is $\Sigma' = \Sigma \times \mathbb{B}^{R_s} \times R_s \times \mathbb{B}^{R_s}$. Let $m = |\Sigma|$. Then, $|\Sigma'| = m \cdot 2^{O(k_s)}$. Thus, the new factor in the complexity is $|\Sigma|$, which is typically much smaller than N. It follows that the synthesis problem for A' can be solved in time $(Nmc)^{O(Nc)} = \big(cmn(k_A + k_s)^{k_A + k_s}\big)^{O\left(cn(k_A + k_s)^{k_A + k_s}\right)}$. Thus, a naive analysis gives a complexity that is doubly-exponential in k_A and k_s and is exponential in n and c. As we argue below, the analysis can be tightened to a one that is doubly-exponential only in k_A and is exponential in n, c, and k_s. Essentially, this follows from the fact that while the partition-component in the state space of A' behaves universally with respect to the registers in R_A, it is deterministic with respect to these in R_s. Consequently, when counting the number of states in the DPW obtained by determinizing A', we can replace the number of all possible partitions of R by the number of partitions of R for a fixed partition of R_s. For more details, see [23].

▶ **Theorem 8.** *Register-bounded synthesis with k_s system registers for reg-UPWs with n states, finite alphabet of size m, index c, and k_A registers, is solvable in time $(cmn(k_s + k_A))^{O(cnk_A(k_s + k_A)^{k_A})}$. Thus, it is polynomial in m, exponential in n, c, and k_s, and doubly-exponential in k_A.*

We note that when the specification automaton A is a reg-UCW, its abstraction A' is a UCW. Since reg-UCWs can be expressed as reg-UPWs with $c = 2$, the obtained time complexity for the case where specifications are reg-UCWs is $(mn(k_s + k_A))^{O(nk_A(k_s + k_A)^{k_A})}$.

4 Synthesis with a Fixed Number of System and Environment Registers

In this section, we consider the system-bounded synthesis problem with respect to restricted environments. Such environments are expressible by a register transducer with a bounded number of registers. Clearly, restricting the environments makes more specifications realizable. As we shall see, however, the complexity of the synthesis problem increases. An important conceptual difference between the setting studied in Section 3 and the one here is that once we fix the number of registers of both the system and the environment, we also fix the number of data values that may participate in the interaction. Indeed, the only data outputs that the system and environment transducers may generate during the interaction are these stored in their registers in their initial valuations.

In order to define the bounded setting, we first have to define the interaction between system and environment transducers. Consider a system transducer $T_{sys} = \langle \Sigma_I, \Sigma_O, S_s, s_0^s, R_s, \nu_0^s, \tau_s \rangle$ and an environment transducer $T_{env} = \langle \Sigma_O, \Sigma_I, S_e, s_0^e, R_e, \nu_0^e, \tau_e \rangle$. Note that the outputs of the environments are the inputs of the system, and vice versa. We denote the computation that is the interaction between the two transducers by $T_{env} \| T_{sys}$, indicating that the environment initiates the interaction and is the first transducer to move. Recall that $\tau_e : S_e \times (\Sigma_O \times \mathbb{B}^{R_e}) \to S_e \times \mathbb{B}^{R_e} \times \Sigma_I \times R_e$. The Σ_O and $\mathbb{B}^{R_e}$ components of the transition depends on the output of the system, which are generated when the system moves between states. Likewise, $\tau_s : S_s \times (\Sigma_I \times \mathbb{B}^{R_s}) \to S_s \times \mathbb{B}^{R_s} \times \Sigma_O \times R_s$, with the Σ_I and $\mathbb{B}^{R_s}$ components depending on the output of the environment. Recall that we assume that the environment moves first. Accordingly, for the first step of the interaction we assume that the Σ_O and $\mathbb{B}^{R_e}$ components are induced by the pair $\langle \varnothing, \nu_0(r_0) \rangle$, for some designated register $r_0 \in R_e$.

Formally, $T_{env} \| T_{sys} = \langle \langle i_0, o_0 \rangle, i_0, o_0 \rangle \langle \langle i_1, o_1 \rangle, i_1, o_1 \rangle \ldots \in ((\Sigma_I \times \Sigma_O) \times \mathcal{D} \times \mathcal{D})^\omega$ is such that there are runs $\rho_e = \langle s_0^e, \nu_0^e \rangle \langle s_1^e, \nu_1^e \rangle \langle s_2^e, \nu_2^e \rangle \ldots \in (S_e \times \mathcal{D}^{R_e})^\omega$ of T_{env} and $\rho_s = \langle s_0^s, \nu_0^s \rangle \langle s_1^s, \nu_1^s \rangle \langle s_2^s, \nu_2^s \rangle \ldots \in (S_s \times \mathcal{D}^{R_s})^\omega$ of T_{sys} such that the following hold. Let $\langle o_{-1}, o_{-1} \rangle = \langle \varnothing, \nu_0^e(r_0^e) \rangle$. Then, for every $j \geq 0$, the following hold:

- $\tau^e(s_j^e, o_{j-1}, \nu_j^e \sim o_{j-1}) = \langle s_{j+1}^e, a_j^e, i_j, r_j^e \rangle$, $i_j = \nu_j^e(r_j^e)$, and $\nu_{j+1}^e = \mathit{update}(\nu_j^e, o_{j-1}, a_j^e)$. That is, in each round in the interaction, including the first round, the environment moves first, the configuration $\langle s_{j+1}^e, \nu_{j+1}^e \rangle$ is the $\langle o_{j-1}, o_{j-1} \rangle$-successor of $\langle s_j^e, \nu_j^e \rangle$, and the transition taken in this move fixes i_j and i_j.
- $\tau^s(s_j^s, i_j, \nu_j^s \sim i_j) = \langle s_{j+1}^s, a_j^s, o_j, r_j^s \rangle$, $o_j = \nu_j^s(r_j^s)$, and $\nu_{j+1}^s = \mathit{update}(\nu_j^s, i_j, a_j^s)$. That is, the system respond by moving to the configuration $\langle s_{j+1}^s, \nu_{j+1}^s \rangle$, which is the $\langle i_j, i_j \rangle$-successor of $\langle s_j^s, \nu_j^s \rangle$, and the transition taken in this move fixes o_j and o_j.

The *environment-system-bounded realizability problem* is to decide, given a reg-UPW A over $\Sigma_I \times \Sigma_O \times \mathcal{D} \times \mathcal{D}$, and numbers k_s and k_e of system and environment registers, respectively, whether there is a system transducer T_{sys} with at most k_s registers such that for all environment transducers T_{env} with at most k_e registers, we have that $T_{env} \| T_{sys} \models A$. The *environment-system-bounded synthesis problem* is to construct such a system transducer, if exists.

Let $A = \langle \Sigma, Q, q_0, R_A, \nu_0^A, \delta, \alpha \rangle$. As in the construction in Section 3, we define a (register-less) UPW A' that abstracts the registers of A and maintains instead the equivalence relation between the registers. Here, however, the equivalence relation refers to the registers of A, of the system, and of the environment. Let R_s and R_e denote the sets of system and environment registers, respectively. Let $R = R_s \cup R_e \cup R_A$, Π be the set of equivalence relations over R, and $f : \mathcal{D}^R \to \Pi$ map a register valuation to the partition it induces. We modify the function update' from Section 3 to refer to registers directly, namely $\mathit{update}' : \Pi \times R \times \mathbb{B}^R \to \Pi$ maps $\langle \pi, r, a \rangle$ to the partition resulting from moving the registers in a into the equivalence class of r. Formally, $\mathit{update}'(\pi, r, a) = \{S \setminus a : S \in \pi \setminus C\} \setminus \{\varnothing\} \cup \{C \cup a\}$, where $C \in \pi$ and $r \in C$. The update function has properties similar to these stated in Lemma 5.

▶ **Lemma 9.** *For every valuation $\nu \in \mathcal{D}^R$, register $r \in R$, and storing mask $a \subseteq R$, we have that $f(\mathit{update}(\nu, \nu(r), a)) = \mathit{update}'(f(\nu), r, a)$.*

Given a reg-UPW A, bounds $k_s, k_e \in \mathbb{N}$, and an initial partition $\pi_0 \in \mathcal{D}^R$, the (k_s, k_e, π_0)-*abstraction* of A is the UPW $A' = \langle \Sigma', Q', q'_0, \delta', \alpha' \rangle$, defined as follows.

- $\Sigma' = \Sigma \times R_s \times \mathbb{B}^{R_s} \times \mathbb{B}^{R_s}$.
- $Q' = (Q \times \Pi \times R_s) \cup \{q'_0\}$. A state $\langle q, \pi, r_s \rangle \in Q \times \Pi \times R_s$ contains, in addition to the original state q and partition π, the register r_s whose value was output by the system transducer in the previous move.
- The initial state $q'_0 = \langle q_0, \pi_0, r^e_0 \rangle$. It contains the environment register r^e_0, because in the first move the environment transducer reads its own data value $\mathfrak{v}^e_0(r^e_0)$.
- Defining δ', we use two auxiliary partitions: First, $\pi^\star$ corresponds to the register valuation after the environment transducer moves and updates its registers. Then, $\pi^{\star\star}$ corresponds to the register valuations after the system transducer moves and updates its registers. Finally, the destination partition π' corresponds to the register valuation after A moves. For every state $\langle q, \pi, r \rangle \in (Q \times \Pi \times R_s) \cup \{\langle q_0, \pi_0, r^e_0 \rangle\}$ and letter $\langle \sigma, r_s, g^s_i, a^s_i \rangle \in \Sigma'$, we have that $\langle q', \pi', r_s \rangle \in \delta'(\langle q, \pi, r \rangle, \langle \sigma, r_s, g^s_i, a^s_i \rangle)$ iff there exist $r_e \in R_e$, $a^e_o \in \mathbb{B}^{R_e}$, and $a^A_i \in \mathbb{B}^{R_A}$ satisfying the following.
  - Let $\pi^\star = \mathit{update}'(\pi, r, a^e_o)$. That is, the environment transducer updates its registers using the previous system value. (In the initial state, the environment transducer uses the value stored in its register r^e_0.)
  - Let $C \in \pi^\star$ be the set that contains r_e. We require that $(C \cap R_s) = g^s_i$.
  - Let $\pi^{\star\star} = \mathit{update}'(\pi^\star, r_e, a^s_i)$. That is, the system transducer updates its registers using the value stored currently in the register that the environment outputs.
  - The automaton A transits and updates its registers using the values in the registers of the environment and system transducers. Hence, the input guard g^A_i is A-chosen by $\langle \pi^{\star\star}, r_e \rangle$, while the output guard g^A_o is A-chosen by $\langle \pi^{\star\star}, r_s \rangle$. Thus, we require that $\langle q', a^A_i \rangle \in \delta(q, \langle \sigma, g^A_i, g^A_o \rangle)$ and $\pi' = \mathit{update}'(\pi^{\star\star}, r_e, a^A_i)$.
- The acceptance condition of A' is induced from the one of A. Thus, for every state $\langle q, \pi, r \rangle \in Q'$, we have that $\alpha'(\langle q, \pi, r \rangle) = \alpha(q)$.

Recall that the abstraction of A is parameterized by both the number of registers that the system transducer may have as well as an initial partition for the registers of the system, the environment, and the automaton. Let $\mathfrak{v}_A \in \mathcal{D}^{R_A}$ be a valuation of the automaton registers, and π_s a partition of R_s. A partition $\pi \in \Pi$ is *consistent with* $\mathfrak{v}_A$ *and* π_s if there are register valuations $\mathfrak{v}_s \in \mathcal{D}^{R_s}$ and $\mathfrak{v}_e \in \mathcal{D}^{R_e}$ s.t. $\pi_s = f(\mathfrak{v}_s)$ and $\pi = f(\mathfrak{v}_A \cup \mathfrak{v}_s \cup \mathfrak{v}_e)$. Thus, automata registers are related according to $\mathfrak{v}^A_0$, system registers are related according to π_s, and environment registers are not related in any special way.

▶ **Theorem 10.** *Consider a reg-UPW A with $\Sigma = \Sigma_I \times \Sigma_O$, set of registers R_A, and an initial valuation $\mathfrak{v}^A_0$. Then, A is realizable by a k_s-register Σ_I/Σ_O-transducer with a set of registers R_s with respect to environments that are k_e-register Σ_O/Σ_I-transducers iff there is a partition π_s of R_s and a $(\Sigma_I \times \mathbb{B}^{R_s})/(\Sigma_O \times R_s \times \mathbb{B}^{R_s})$-transducer T' such that for every partition π_0 of R that is consistent with $\mathfrak{v}^A_0$ and π_s, the transducer T' realizes the (k_s, k_e, π_0)-abstraction of A.*

Proof sketch. The theorem follows from the following claim, which we prove in [23]. Fix a system k_s-register transducer T_{sys} with an initial valuation $\mathfrak{v}^s_0$, and fix an environment initial valuation $\mathfrak{v}^e_0$. Let A' be the (k_s, k_e, π_0)-abstraction of A with $\pi_0 = f(\mathfrak{v}^s_0 \cup \mathfrak{v}^e_0 \cup \mathfrak{v}^A_0)$. Let T'_{sys} be the register-less transducer corresponding to T_{sys}. Then, we have that $T'_{sys} \models A'$ iff for every environment transducer T_{env} with the initial valuation $\mathfrak{v}^e_0$, it holds that $T_{env} \| T_{sys} \models A$. ◀

We now analyze the complexity of the environment-system-bounded synthesis problem. Using Theorem 10, we can reduce the synthesis problem for k_s system and k_e environment registers, reg-UPW A with n states, index c, and k_A registers, to the synthesis problem of a (register-less) UPW A' with $O(nk^k)$ states and index c, where $k = k_s + k_e + k_A$. Recall that the reduction does not create a single instance of the register-less synthesis problem, and instead requires to find a system partition π_s such that the (k_s, k_e, π_0)-abstractions of A, for every π_0 consistent with v_0^A and π_s, are realized by a single transducer. There can be no more than $k_s^{k_s}$ system partitions, and we are going to enumerate them one by one. Now, once a system partition π_s is fixed, we can create a single UPW that represents the intersection of the abstraction UPWs for each π_0 consistent with π_s and v_0^A. To this end, we create one initial state per π_0, while the rest of the definition stays the same. The number of initial states is bounded by $(k_s + k_e + k_A)^{k_e}$. Let us call this automaton A'. By the same naive analysis as in the system-bounded case, the synthesis problem for A' can be solved in time $(Nmc)^{O(Nc)} = \left(cmnk^k\right)^{O\left(cnk^k\right)}$, where $m = |\Sigma|$ is the size of the finite alphabet of A. In order to account for enumeration of system partitions, we multiply it by $k_s^{k_s}$, but this does not affect the asymptotic complexity. Thus, the environment-system-bounded synthesis problem is doubly-exponential in k_A, k_s, and k_e, and is exponential in n and c.

As in the case of system-bounded synthesis, we can use the fact that the system-partition component in the state space of A' is deterministic with respect to the registers in R_s, and behaves universally only with respect to the registers in R_A and R_e. The universal behavior with respect to R_e follows from the fact that a system transducer plays against all possible environment transducers. Accordingly, we can tighten the complexity as follows.

▶ **Theorem 11.** *Environment-system-bounded synthesis with k_s system and k_e environment registers for reg-UPWs with n states, finite alphabet of size m, index c, and k_A registers is solvable in time $(cmn(k_s + k_e + k_A))^{O(cn(k_s+k_e+k_A)^{(k_e+k_A+1)})}$. Thus, it is polynomial in m, exponential in c, n, and k_s, and doubly-exponential in k_A and k_e.*

References

1 R. Bloem, K. Chatterjee, and B. Jobstmann. Graph Games and Reactive Synthesis. In *Handbook of Model Checking.*, pages 921–962. Springer, 2018.

2 M. Bojańczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. *Journal of the ACM*, 56(3):1–48, 2009.

3 M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-Variable Logic on Words with Data. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 7–16, 2006.

4 A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *FCT*, pages 1–22, 2007.

5 A. Bouajjani, P. Habermehl, and R R. Mayr. Automatic verification of recursive procedures with one integer parameter. *Theoretical Computer Science*, 295:85–106, 2003.

6 M. Brambilla, S. Ceri, S. Comai, P. Fraternali, and I. Manolescu. Specification and Design of Workflow-Driven Hypertexts. *J. Web Eng.*, 1(2):163–182, 2003.

7 C.S. Calude, S. Jain, B. Khoussainov, W. Li, and F. Stephan. Deciding parity games in quasipolynomial time. In *Proc. 49th ACM Symp. on Theory of Computing*, pages 252–263, 2017.

8 S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera. *Designing Data-Intensive Web Applications.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

9 K. Chatterjee, T. Henzinger, and B. Jobstmann. Environment Assumptions for Synthesis. In *Proc. 19th Int. Conf. on Concurrency Theory*, volume 5201 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2008.

10 A. Church. Logic, arithmetics, and automata. In *Proc. Int. Congress of Mathematicians, 1962*, pages 23–35. Institut Mittag-Leffler, 1963.

11 G. Delzanno, A. Sangnier, and R. Traverso. Parameterized Verification of Broadcast Networks of Register Automata. In P. A. Abdulla and I. Potapov, editors, *Reachability Problems*, pages 109–121, Berlin, Heidelberg, 2013. Springer.

12 S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3):16:1–16:30, 2009.

13 R. Ehlers. Symbolic bounded synthesis. In *Proc. 22nd Int. Conf. on Computer Aided Verification*, volume 6174 of *Lecture Notes in Computer Science*, pages 365–379. Springer, 2010.

14 R. Ehlers, S. Seshia, and H. Kress-Gazit. Synthesis with Identifiers. In *Proc. 15th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 8318 of *Lecture Notes in Computer Science*, pages 415–433. Springer, 2014.

15 L. Exibard, E. Filiot, and P-A. Reynier. Synthesis of Data Word Transducers. In *Proc. 30th Int. Conf. on Concurrency Theory*, 2019.

16 E. Filiot, N. Jin, and J.-F. Raskin. An Antichain Algorithm for LTL Realizability. In *Proc. 21st Int. Conf. on Computer Aided Verification*, volume 5643, pages 263–277, 2009.

17 B. Finkbeiner, F. Klein, R. Piskac, and M. Santolucito. Temporal Stream Logic: Synthesis beyond the Bools. In *Proc. 31st Int. Conf. on Computer Aided Verification*, 2019.

18 O. Grumberg, O. Kupferman, and S. Sheinvald. Variable Automata over Infinite Alphabets. In *Proc. 4th Int. Conf. on Language and Automata Theory and Applications*, volume 6031 of *Lecture Notes in Computer Science*, pages 561–572. Springer, 2010.

19 O. Grumberg, O. Kupferman, and S. Sheinvald. An Automata-Theoretic Approach to Reasoning about Parameterized Systems and Specifications. In *11th Int. Symp. on Automated Technology for Verification and Analysis*, pages 397–411, 2013.

20 R. Hojati, D.L. Dill, and R.K. Brayton. Verifying linear temporal properties of data insensitive controllers using finite instantiations. In *Hardware Description Languages and their Applications*, pages 60–73. Springer, 1997.

21 M. Kaminski and N. Francez. Finite-memory automata. *Theoretical Computer Science*, 134(2):329–363, 1994.

22 M. Kaminski and D. Zeitlin. Extending finite-memory automata with non-deterministic reassignment. In *AFL*, pages 195–207, 2008.

23 A. Khalimov and O. Kupferman. Register-bounded Synthesis, 2019. Full version, available on the author's personal pages.

24 A. Khalimov, B. Maderbacher, and R. Bloem. Bounded Synthesis of Register Transducers. In *16th Int. Symp. on Automated Technology for Verification and Analysis*, volume 11138 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2018.

25 O. Kupferman. Recent Challenges and Ideas in Temporal Synthesis. In *Proc. 38th International Conference on Current Trends in Theory and Practice of Computer Science*, volume 7147 of *Lecture Notes in Computer Science*, pages 88–98. Springer, 2012.

26 O. Kupferman, Y. Lustig, M.Y. Vardi, and M. Yannakakis. Temporal Synthesis for Bounded Systems and Environments. In *Proc. 28th Symp. on Theoretical Aspects of Computer Science*, pages 615–626, 2011.

27 R. Lazić and D. Nowak. A Unifying Approach to Data-Independence. In *Proc. 11th Int. Conf. on Concurrency Theory*, pages 581–596. Springer Berlin Heidelberg, 2000.

28 F. Neven, T. Schwentick, and V. Vianu. Towards Regular Languages over Infinite Alphabets. In *26th Int. Symp. on Mathematical Foundations of Computer Science*, pages 560–572. Springer-Verlag, 2001.

29 N. Piterman. From Nondeterministic Büchi and Streett Automata to Deterministic Parity Automata. In *Proc. 21st IEEE Symp. on Logic in Computer Science*, pages 255–264. IEEE press, 2006.

30 A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *Proc. 16th ACM Symp. on Principles of Programming Languages*, pages 179–190, 1989.

31 S. Schewe and B. Finkbeiner. Bounded Synthesis. In *5th Int. Symp. on Automated Technology for Verification and Analysis*, volume 4762 of *Lecture Notes in Computer Science*, pages 474–488. Springer, 2007.

32 S. Schewe and T. Varghese. Determinising Parity Automata. In *39th Int. Symp. on Mathematical Foundations of Computer Science*, volume 8634 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2014.

33 Y. Shemesh and N.: Francez. Finite-state unification automata and relational languages. *Information and Computation*, 114:192–213, 1994.

34 T. Tan. *Pebble Automata for Data Languages: Separation, Decidability, and Undecidability*. PhD thesis, Technion - Computer Science Department, 2009.

35 N. Tzevelekos. Fresh-register Automata. In *Proc. 38th ACM Symp. on Principles of Programming Languages*, pages 295–306, New York, NY, USA, 2011. ACM.

36 V. Vianu. Automatic verification of database-driven systems: a new frontier. In *ICDT '09*, pages 1–13, 2009.

37 P. Wolper. Expressing Interesting Properties of Programs in Propositional Temporal Logic. In *Proc. 13th ACM Symp. on Principles of Programming Languages*, pages 184–192, 1986.

Translating Asynchronous Games for Distributed Synthesis

Raven Beutner
Saarland University, Saarbrücken, Germany

Bernd Finkbeiner
Saarland University, Saarbrücken, Germany

Jesko Hecking-Harbusch
Saarland University, Saarbrücken, Germany

Abstract

In distributed synthesis, a set of process implementations is generated, which together, accomplish an objective against all possible behaviors of the environment. A lot of recent work has focussed on systems with causal memory, i.e., sets of asynchronous processes that exchange their causal histories upon synchronization. Decidability results for this problem have been stated either in terms of control games, which extend Zielonka's asynchronous automata by partitioning the actions into controllable and uncontrollable, or in terms of Petri games, which extend Petri nets by partitioning the tokens into system and environment players. The precise connection between these two models was so far, however, an open question.

In this paper, we provide the first formal connection between control games and Petri games. We establish the equivalence of the two game types based on weak bisimulations between their strategies. For both directions, we show that a game of one type can be translated into an equivalent game of the other type. We provide exponential upper and lower bounds for the translations. Our translations allow to transfer and combine decidability results between the two types of games. Exemplarily, we translate decidability in acyclic communication architectures, originally obtained for control games, to Petri games, and decidability in single-process systems, originally obtained for Petri games, to control games.

2012 ACM Subject Classification Theory of computation → Algorithmic game theory

Keywords and phrases synthesis, distributed systems, causal memory, Petri games, control games

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.26

Related Version A full version of the paper is available at `https://arxiv.org/abs/1907.00829` [3].

Funding Supported by the German Research Foundation (DFG) Grant Petri Games (392735815) and the Collaborative Research Center "Foundations of Perspicuous Software Systems" (TRR 248, 389792660), and by the European Research Council (ERC) Grant OSARES (683300).

1 Introduction

Synthesis is the task of automatically generating an implementation fulfilling a given objective or proving that no such implementation can exist. Synthesis can be viewed as a game between the *system* and the *environment* with winning strategies for the system being correct implementations [4]. We call a class of games *decidable* if we can determine the existence of a winning strategy. A distributed system consists of local processes, that possess incomplete information about the global system state. *Distributed synthesis* searches for distributed strategies that govern the local processes such that the system as a whole satisfies an objective, independently of the inputs that are received from the environment.

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 26; pp. 26:1–26:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

After some early results on *synchronous* distributed systems [24], most work has focussed on the synthesis of *asynchronous* distributed systems with *causal memory* [12, 13, 20, 14, 11, 10]. Causal memory means that two processes share no information while they run independently; during every synchronization, however, they exchange their complete local histories. The study of the synthesis problem with causal memory has, so far, been carried out, independently of each other, in two different models: control games and Petri games.

Control Games and Petri Games. *Control games* [13] are based on Zielonka's asynchronous automata [26], which are compositions of local processes. The actions of the asynchronous automaton are partitioned as either controllable or uncontrollable. Hence, each process can have both controllable and uncontrollable behavior. A strategy comprises a family of one individual controller for each process that can restrict controllable actions based on the causal past of the process but has to account for all uncontrollable actions. Together, the local controllers aim to fulfill an objective against all possible unrestricted behavior. There are non-elementary decidability results for acyclic communication architectures [13, 20]. Decidability has also been obtained for restrictions on the dependencies of actions [12] or on the synchronization behavior [16, 17] and, recently, for decomposable games [14].

Petri games [11] are based on Petri nets. They partition the places of the underlying Petri net into system places and environment places and, thereby, group the tokens into system players and environment players. For tokens in system places, the outgoing transitions can be restricted by the strategy whereas tokens in environment places cannot be controlled, i.e., every possible transition has to be accounted for. Strategies are defined as restrictions of the global unfolding and aim to fulfill an objective against all possible unrestricted behavior. Petri games are EXPTIME-complete for a bounded number of system players and one environment player [11] as well as for one system player and a bounded number of environment players [10]. Both models are based on causal information: Control games utilize local views whereas Petri games utilize unfoldings.

Translations. The precise connection between control games and Petri games, and hence, the question whether results can be transferred between them, was, so far, open. We translate control games into Petri games, and vice versa. Both game types admit strategies based on causal information but the formalisms for the possibilities of system and environment differ. In control games, an action is either controllable or uncontrollable and therefore can be restricted by either all or none of the involved players. From the same state of a process, both controllable and uncontrollable behavior is possible. By contrast, Petri games utilize a partitioning into system and environment places. While this offers more precise information about which player can control a shared transition, a given place can no longer comprise both system and environment behavior. The challenge is to resolve the controllability while preserving the causal information in the game. For both translations, we adopt the concept of *commitment sets*: The local players do not enable behavior directly but move to a state or place that explicitly encodes their decision of what to enable. Using this explicit representation, we can express the controllability aspects of one game in the respective other one, i.e., make actions in a control game controllable by only a subset of players and allow places in Petri games that comprise both environment and system behavior.

Our translations preserve the structure of winning strategies in a weak bisimilar way. In addition to the upper bounds established by our exponential translations, we provide matching lower bounds. The translations show that contrasting formalisms can be overcome whereas our lower bounds highlight an intrinsic difficultly to achieve this. The equivalence

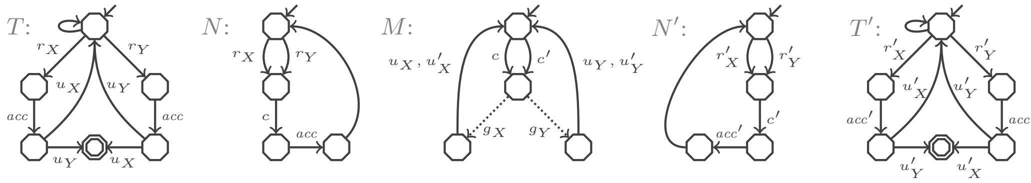
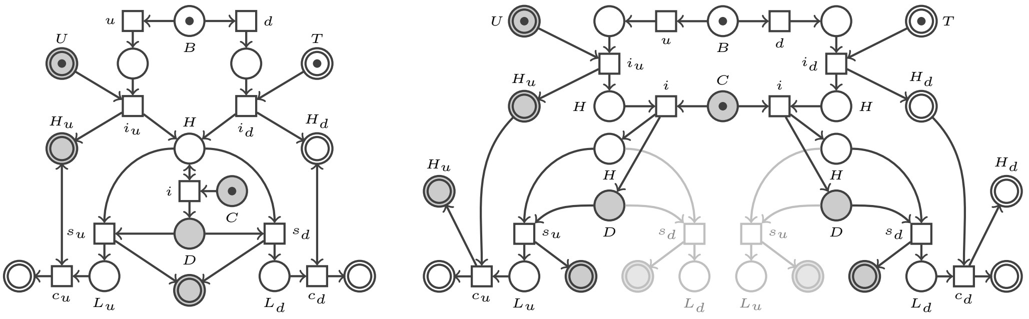


Figure 1 A control game for a manager M of resources X and Y between threads T and T' with networks N and N' is depicted. Communication occurs by synchronization on shared actions. Dotted actions are controllable, all others are uncontrollable. Losing states are double circles.



(a) Petri game for a police strategy. **(b)** Unfolding (with grayed parts) and winning strategy (without).

Figure 2 A Petri game, an unfolding, and a winning strategy are given. Gray places belong to the system whereas white places belong to the environment. Winning places are double circles.

of both models, as witnessed by our results, gives rise to more practical applications by allowing the transfer of existing decidability results between both models. As an example, we can transfer decidability of single-process systems for Petri games [10] to control games and decidability for acyclic communication architectures for control games [13] to Petri games.

2 Examples

We illustrate the models with two examples. The examples demonstrate the use of control games and Petri games and their differences, which our translations overcome. Both examples highlight decidable classes [10, 13], that are transferable through the results of the paper.

As a control game, consider the example of a manager for resources in Fig. 1. The control game consists of five players: A manager M and two pairs of thread and network connection (T, N and T', N'). Both pairs of thread and network connection are identical but act on disjoint actions (primed and not). There are two resources X and Y that are managed by M. Each thread (T, T') can request access to one of them (r_X, r_Y) and afterwards wait for the acknowledgement from its network connection (acc). After the acknowledgement, the thread can use one of the resources (u_X, u_Y). Each network connection (N, N') synchronizes with its thread on the actions for requests and synchronizes with the manager for communication (c). Afterwards, each network connection sends the acknowledgement to its thread. The manager is the only process that comprises controllable actions. Upon communication with one of the two network connections, the manager can grant access to the resources X or Y using the controllable actions g_X or g_Y. The enabled resource can afterwards be accessed and used (u_X, u_Y). A losing state can be reached for either thread if an unwanted resource is enabled, i.e., after the acknowledgement, the requested and granted resource do not match.

This control game can be won by the system. After every communication with a network connection, the manager enables the resource that the respective thread requested. A winning controller relies on the information transfer associated with every synchronization. The request of the process is transferred to the manager upon communication with the network connection. Then, the correct resource can be enabled. This control game falls into a decidable class by our translation to Petri games as it is a single-process system with bad places [10]. Note that the control game has a cyclic communication architecture.

As a Petri game, consider the example of a burglary in Fig. 2a. A crime boss in environment place B decides to either burgle up- or downtown by firing transition u or d. Depending on the choice, an undercover agent in system place U or a thug in environment place T is instructed by transition i_u or i_d and commits the burglary, i.e., moves to place H_u or H_d. This returns the crime boss to her hideout H where she gets caught and interrogated (i) by a cop in system place C. Afterwards, the cop can send (s_u, s_d) the flipped crime boss up- or downtown to place L_u or L_d in order to intercept the burglary (c_u, c_d).

Causal past is key for the existence of winning strategies. Only upon synchronization players exchange all information about their past. After the crime boss instructs for a location to burgle, only she and the respective burglar know about the decision. The cop learns about the location of the burglary after catching the crime boss. A winning strategy for the cop catches and interrogates the crime boss and then uses the obtained information to send the flipped crime boss to the correct location. For this Petri game, our translation results in a control game with acyclic communication architecture [13]. Note that the Petri game has two system and two environment players.

3 Background

We recall asynchronous automata [26], control games [13], Petri nets [25], and Petri games [11].

3.1 Zielonka's Asynchronous Automata

An *asynchronous automaton* [26] is a family of finite automata, called *processes*, synchronizing on shared actions. Our definitions follow [13]. The finite set of processes of an asynchronous automaton is defined as $\mathfrak{P}$. A *distributed alphabet* (Σ, dom) consists of a finite set of *actions* Σ and a *domain* function $dom : \Sigma \to 2^{\mathfrak{P}} \setminus \{\emptyset\}$. For an action $a \in \Sigma$, $dom(a)$ are all processes that have to synchronize on a. For a process $p \in \mathfrak{P}$, $\Sigma_p = \{a \in \Sigma \mid p \in dom(a)\}$ denotes all actions p is involved in. A (deterministic) asynchronous automaton $\mathcal{A} = (\{S_p\}_{p\in\mathfrak{P}}, s_{in}, \{\delta_a\}_{a\in\Sigma})$ is defined by a finite set of local states S_p for every process $p \in \mathfrak{P}$, the initial state $s_{in} \in \prod_{p\in\mathfrak{P}} S_p$, and a partial function $\delta_a : \prod_{p\in dom(a)} S_p \dashrightarrow \prod_{p\in dom(a)} S_p$. We call an element $\{s_p\}_{p\in\mathfrak{P}} \in \prod_{p\in\mathfrak{P}} S_p$ *a global state*. For a set of processes $R \subseteq \mathfrak{P}$, we abbreviate $s_R = \{s_p\}_{p\in R}$ as the restriction of the global state to R. We denote that a local state $s' \in S_p$ is part of a global state s_R by $s' \in s_R$. For a local state s', we define the set of outgoing actions by $act(s') = \{a \in \Sigma \mid \exists s_{dom(a)} \in domain(\delta_a) : s' \in s_{dom(a)}\}$. We can view an asynchronous automaton as a sequential automaton with state space $\prod_{p\in\mathfrak{P}} S_p$ and transitions $s \xrightarrow{a} s'$ if $(s_{dom(a)}, s'_{dom(a)}) \in \delta_a$ and $s_{\mathfrak{P}\setminus dom(a)} = s'_{\mathfrak{P}\setminus dom(a)}$. By $Plays(\mathcal{A}) \in \Sigma^* \cup \Sigma^\omega$, we denote the set of finite and infinite sequences in this global automaton. For a finite $u \in Plays(\mathcal{A})$, $state(u)$ denotes the global state after playing u and $state_p(u)$ the local state of process p.

The domain function dom induces an independence relation I: Two actions $a, b \in \Sigma$ are independent, denoted by $(a, b) \in I$, if they involve different processes, i.e., $dom(a) \cap dom(b) = \emptyset$. Adjoint independent actions of sequences of actions can be swapped. This leads to an equivalence relation $\sim_I$ between sequences, where $u \sim_I w$ if u and w are identical up to multiple swaps of consecutive independent actions. The equivalence classes of $\sim_I$ are called *traces* and denoted by $[u]_I$ for a sequence u. Given the definition of asynchronous automata, it is natural to abstract from concrete sequences and consider $\mathit{Plays}(\mathcal{A})$ as a set of traces.

In our translation, an alternative characterization of a subset of asynchronous automata turns out to be practical: We describe every process p by a finite local automaton $\mathfrak{L}_p = (Q_p, s_{0,p}, \vartheta_p)$ acting on actions from Σ_p. Here, Q_p is a finite set of states, $s_{0,p}$ the initial state and $\vartheta_p \subseteq Q_p \times \Sigma_p \times Q_p$ a deterministic transition relation. For a family of local processes $\{\mathfrak{L}_p\}_{p\in\mathfrak{P}}$, we define the *parallel composition* $\bigotimes_{p\in\mathfrak{P}} \mathfrak{L}_p$ as an asynchronous automaton with **(1)** $\forall p \in \mathfrak{P} : S_p = Q_p$, **(2)** $s_{in} = \{s_{0,p}\}_{p\in\mathfrak{P}}$, and **(3)** $\delta_a(\{s_p\}_{p\in dom(a)})$: If for all $p \in dom(a)$, there exists a state $s'_p \in S_p$ with $(s_p, a, s'_p) \in \vartheta_p$ then define $\delta_a(\{s_p\}_{p\in dom(a)}) = \{s'_p\}_{p\in dom(a)}$, otherwise it is undefined. Figure 1 is an example of such a parallel composition. Note that not every asynchronous automaton can be described as a composition of local automata.

3.2 Control Games

A *control game* [13] $\mathcal{C} = (\mathcal{A}, \Sigma^{sys}, \Sigma^{env}, \{\mathcal{S}_p\}_{p\in\mathfrak{P}})$ consists of an asynchronous automaton $\mathcal{A}$ as a game arena, a distribution of actions into *controllable* actions Σ^{sys} and *uncontrollable* actions Σ^{env}, and *special states* $\{\mathcal{S}_p\}_{p\in\mathfrak{P}}$ for a winning objective. We define the set of plays in the game as $\mathit{Plays}(\mathcal{C}) = \mathit{Plays}(\mathcal{A})$. Intuitively, a strategy for $\mathcal{C}$ can restrict controllable actions but cannot prohibit uncontrollable actions. Given a play u, a process p only observes parts of it. The local p-view, denoted by $\mathit{view}_p(u)$, is the shortest trace $[v]_I$ such that $u \sim_I v\,w$ for some w not containing any actions from Σ_p. The p-view describes the *causal past* of process p and contains all actions the process is involved in and all actions it learns about via communication. We define the set of p-views as $\mathit{Plays}_p(\mathcal{C}) = \{\mathit{view}_p(u) \mid u \in \mathit{Plays}(\mathcal{C})\}$.

To avoid confusion with Petri games, we refer to strategies for control games as controllers. A *controller* for $\mathcal{C}$ is a family of local controllers for all processes $\varrho = \{f_p\}_{p\in\mathfrak{P}}$. A local controller for a process p is a function $f_p : \mathit{Plays}_p(\mathcal{C}) \to \Sigma^{sys} \cap \Sigma_p$. $\mathit{Plays}(\mathcal{C}, \varrho)$ denotes the set of plays respecting ϱ. It is defined as the smallest set containing the empty play ϵ and such that for every $u \in \mathit{Plays}(\mathcal{C}, \varrho)$: **(1)** if $a \in \Sigma^{env}$ and $ua \in \mathit{Plays}(\mathcal{C})$ then $ua \in \mathit{Plays}(\mathcal{C}, \varrho)$ and **(2)** if $a \in \Sigma^{sys}$, $ua \in \mathit{Plays}(\mathcal{C})$, and $\forall p \in dom(a) : a \in f_p(\mathit{view}_p(u))$ then $ua \in \mathit{Plays}(\mathcal{C}, \varrho)$. Environment actions are always possible whereas system actions are only possible if allowed by the local controllers of *all* participating processes. Local controllers base their decisions on their local view and thereby act only on their causal past.

We define the (possibly empty) set of *final plays* $\mathit{Plays}^F(\mathcal{C}, \varrho)$ as all finite plays $u \in \mathit{Plays}(\mathcal{C}, \varrho)$ such that there is no a with $u\,a \in \mathit{Plays}(\mathcal{C}, \varrho)$. We consider either reachability or safety objectives for the system. Therefore, $\{\mathcal{S}_p\}_{p\in\mathfrak{P}}$ describes sets of winning ($\mathcal{W}_p$) or losing ($\mathcal{B}_p$) states. A controller ϱ is *reachability-winning* if it only admits finite plays and on each final play all processes terminate in a winning state. For safety objectives, we need to ensure progress. A controller ϱ is *deadlock-avoiding* if $\mathit{Plays}^F(\mathcal{C}, \varrho) \subseteq \mathit{Plays}^F(\mathcal{C}, \top)$ for the controller $\top$ allowing all actions, i.e., the controller only terminates if the asynchronous automaton does. A controller ϱ is *safety-winning* if it is deadlock-avoiding and no play in $\mathit{Plays}(\mathcal{C}, \varrho)$ visits any local, losing state from $\bigcup_{p\in\mathfrak{P}} \mathcal{B}_p$.

3.3 Petri Nets

A *Petri net* [25, 22] $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \mathit{In})$ consists of disjoint sets of *places* $\mathcal{P}$ and *transitions* $\mathcal{T}$, the *flow relation* $\mathcal{F}$ as multiset over $(\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$, and the *initial marking* In as multiset over $\mathcal{P}$. We call elements in $\mathcal{P} \cup \mathcal{T}$ *nodes* and $\mathcal{N}$ *finite* if the set of nodes is finite. For node x, the *precondition* (written $\mathit{pre}(x)$) is the multiset defined by $\mathit{pre}(x)(y) = \mathcal{F}(y, x)$ and *postcondition* (written $\mathit{post}(x)$) the multiset defined by $\mathit{post}(x)(y) = \mathcal{F}(x, y)$. For multiple nets $\mathcal{N}^\sigma, \mathcal{N}^1, \cdots$, we refer to the components by $\mathcal{P}^{\mathcal{N}^\sigma}$ and write $\mathit{pre}^{\mathcal{N}^\sigma}(x)$ unless clear from the context. Configurations of Petri nets are represented by multisets over places, called *markings*. *In* is the initial marking. For a transition t, $\mathit{pre}(t)$ is the multiset of places from which tokens are consumed. A transition t is *enabled* in marking M if $\mathit{pre}(t) \subseteq M$, i.e., every place in M contains at least as many tokens as required by t. If no transition is enabled from marking M then we call M *final*. An enabled transition t can *fire* from a marking M resulting in the successor marking $M' = M - \mathit{pre}(t) + \mathit{post}(t)$ (denoted $M\ [\,t\,\rangle\ M'$). For markings M and M', we write $M\ [\,t_0,\ldots,t_{n-1}\,\rangle\ M'$ if there exist markings $M = M_0, \ldots, M_n = M'$ s.t. $M_i\ [\,t_i\,\rangle\ M_{i+1}$ for all $0 \leq i \leq n-1$. The set of *reachable markings* of $\mathcal{N}$ is defined as $\mathcal{R}(\mathcal{N}) = \{M \mid \exists n \in \mathbb{N}, t_0, \ldots, t_{n-1} \in \mathcal{T} : \mathit{In}\ [\,t_0,\ldots,t_{n-1}\,\rangle\ M\}$. A net $\mathcal{N}'$ is a *subnet* of $\mathcal{N}$ (written $\mathcal{N}' \sqsubseteq \mathcal{N}$) if $\mathcal{P}' \subseteq \mathcal{P}$, $\mathcal{T}' \subseteq \mathcal{T}$, $\mathit{In}' \subseteq \mathit{In}$, and $\mathcal{F}' = \mathcal{F} \upharpoonright (\mathcal{P}' \times \mathcal{T}') \cup (\mathcal{T}' \times \mathcal{P}')$. A Petri net is *1-bounded* if every reachable marking contains at most one token per place. It is *concurrency-preserving* if $|\mathit{pre}(t)| = |\mathit{post}(t)|$ for all transitions t.

For nodes x and y, we write $x \lessdot y$ if $x \in \mathit{pre}(y)$, i.e., there is an arc from x to y. With $\leq$, we denote the reflexive, transitive closure of $\lessdot$. The *causal past* of x is $\mathit{past}(x) = \{y \mid y \leq x\}$. x and y are *causally related* if $x \leq y \vee y \leq x$. They are *in conflict* (written $x \sharp y$) if there exists a place $q \in \mathcal{P} \setminus \{x, y\}$ and two distinct transitions $t_1, t_2 \in \mathit{post}(q)$ s.t. $t_1 \leq x$ and $t_2 \leq y$. Node x is in *self-conflict* if $x \sharp x$. We call x and y *concurrent* if they are neither causally related nor in conflict. An *occurrence net* is a Petri net $\mathcal{N}$, where the pre- and postcondition of all transitions are sets, the initial marking coincides with places without ingoing transitions ($\forall q \in \mathcal{P} : q \in \mathit{In} \Leftrightarrow |\mathit{pre}(q)| = 0$), all other places have exactly one ingoing transition ($\forall q \in \mathcal{P} \setminus \mathit{In} : |\mathit{pre}(q)| = 1$), $\leq$ is *well-founded* (no infinite path following the inverse flow relation exists), and no transition is in self-conflict. An *initial homomorphism* from $\mathcal{N}$ to $\mathcal{N}'$ is a function $\lambda : \mathcal{P} \cup \mathcal{T} \to \mathcal{P}' \cup \mathcal{T}'$ that respects node types ($\lambda(\mathcal{P}) \subseteq \mathcal{P}' \wedge \lambda(\mathcal{T}) \subseteq \mathcal{T}'$), is structure-preserving on transitions ($\forall t \in \mathcal{T} : \lambda[\mathit{pre}^{\mathcal{N}}(t)] = \mathit{pre}^{\mathcal{N}'}(\lambda(t)) \wedge \lambda[\mathit{post}^{\mathcal{N}}(t)] = \mathit{post}^{\mathcal{N}'}(\lambda(t))$), and agrees on the initial markings ($\lambda[\mathit{In}] = \mathit{In}'$).

A branching process [5, 18, 6] describes parts of the behavior of a Petri net. Formally, an *(initial) branching process* of a Petri net $\mathcal{N}$ is a pair $\iota = (\mathcal{N}^\iota, \lambda^\iota)$ where $\mathcal{N}^\iota$ is an occurrence net and $\lambda^\iota : \mathcal{P}^\iota \cup \mathcal{T}^\iota \to \mathcal{P} \cup \mathcal{T}$ is an initial homomorphism from $\mathcal{N}^\iota$ to $\mathcal{N}$ that is injective on transitions with the same precondition ($\forall t, t' \in \mathcal{T}^\iota : (\mathit{pre}^{\mathcal{N}^\iota}(t) = \mathit{pre}^{\mathcal{N}^\iota}(t') \wedge \lambda^\iota(t) = \lambda^\iota(t')) \Rightarrow t = t'$). A branching process describes subsets of possible behaviors of a Petri net. Whenever a place or transition can be reached on two distinct paths it is split up. λ can be thought of as label of the copies into nodes of $\mathcal{N}$. The injectivity condition avoids additional unnecessary splits: Each transition must either be labelled differently or occur from different preconditions. The *unfolding* $\mathfrak{U}$ of $\mathcal{N}$ is the maximal branching process: Whenever there is a set of pairwise concurrent places C s.t. $\lambda[C] = \mathit{pre}^{\mathcal{N}}(t)$ for some transition t then there exists t' with $\lambda(t') = t$ and $\mathit{pre}^{\mathcal{N}^{\mathfrak{U}}}(t') = C$. It represents ever possible behavior of $\mathcal{N}$.

3.4 Petri Games

A *Petri game* [11] is a tuple $\mathcal{G} = (\mathcal{P}_\mathcal{S}, \mathcal{P}_\mathcal{E}, \mathcal{T}, \mathcal{F}, \mathit{In}, \mathit{Sp})$. The *system places* $\mathcal{P}_\mathcal{S}$ and *environment places* $\mathcal{P}_\mathcal{E}$ partition the places of the underlying, finite net $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, \mathit{In})$ with $\mathcal{P} = \mathcal{P}_\mathcal{S} \uplus \mathcal{P}_\mathcal{E}$. We extend notation from the underlying net to $\mathcal{G}$ by, e.g., defining $\mathit{pre}^{\mathcal{G}}(\cdot) = \mathit{pre}^{\mathcal{N}}(\cdot)$ and $\mathcal{P}^{\mathcal{G}} = \mathcal{P}^{\mathcal{N}}$. The game progresses by firing transitions in the underlying net. Intuitively, a strategy can control the behavior of tokens on system places by deciding which transitions to allow. Tokens on environment places belong to the environment and cannot be restricted by strategies. $\mathit{Sp} \subseteq \mathcal{P}$ denotes *special places* used to pose a winning objective. For graphical representation, we depict a Petri game as the underlying net and color system places gray, environment places white, and special places as double circles (cf. Fig. 2).

A *strategy* for $\mathcal{G}$ is an initial branching process $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ satisfying *justified refusal*: If there is a set of pairwise concurrent places C in $\mathcal{N}^\sigma$ and a transition $t \in \mathcal{T}^\mathcal{G}$ with $\lambda[C] = pre^\mathcal{G}(t)$ then there either is a transition t' with $\lambda(t') = t$ and $C = pre^{\mathcal{N}^\sigma}(t')$ or there is a system place $q \in C \cap \lambda^{-1}[\mathcal{P}_\mathcal{S}]$ with $t \notin \lambda[post^{\mathcal{N}^\sigma}(q)]$. Since a branching process describes subsets of the behavior of a Petri net, a strategy is a restriction of possible moves in the game. Justified refusal enforces that only system places can prohibit transitions based on their causal past. From every situation in the game, a transition possible in the underlying net is either allowed, i.e., in the strategy, or there is a *system* place that never allows it. In particular, transitions involving only environment places are always possible. A strategy σ is *reachability-winning* for a set of winning places $Sp = \mathcal{W}$ if $\mathcal{N}^\sigma$ is a finite net and in each final, reachable marking every token is on a winning place. A strategy is *deadlock-avoiding* if for every final, reachable marking M in the strategy, $\lambda[M]$ is final as well, i.e., the strategy is only allowed to terminate if the underlying Petri net does so. A strategy σ is *safety-winning* for bad places $Sp = \mathcal{B}$ if it is deadlock-avoiding and no reachable marking contains a bad place. For both objectives, we can require σ to be *deterministic*: For every reachable marking M and system place $q \in M$ there is at most one transition from $post^{\mathcal{N}^\sigma}(q)$ enabled in M. In Fig. 2b, the unfolding of Fig. 2a is depicted labeled by λ. Excluding the grayed parts, this is a winning strategy for the system.

For safety as winning objective, unbounded Petri games are undecidable in general [11] whereas bounded ones with either one system player [10] or one environment player [11] are EXPTIME-complete. Bounded synthesis is a semi-decision procedure to find winning strategies [7, 8, 15]. Both approaches are implemented in the tool ADAM [9, 8].

4 Game Equivalence

A minimum requirement for translations between games is to be *winning-equivalent*. The system has a winning strategy in one game if and only if it has a winning strategy in the translated other one. One trivial translation fulfilling this is to solve the game and to return a minimal winning-equivalent game. Such a translation is not desirable, especially since decidability in both control games and Petri games is still an open question [19, 11]. Instead, our translations preserve the underlying structure of the games. We propose *strategy-equivalence* as an adequate equivalence notion. Our notion is based on weak bisimulation which is popular and powerful to relate concurrent systems represented as Petri nets [2, 1, 23].

For our purpose, a bisimulation between the underlying Petri net and the asynchronous automaton is not sufficient. Instead, we want to express that any strategy can be matched by a strategy that allows equivalent (bisimilar) behavior, i.e., allows identical actions/transitions. In both models, strategies are defined based on the causal past of the players. A Petri game $\mathcal{G}$ utilizes unfoldings whereas a control game $\mathcal{C}$ utilizes local views. We consider a strategy and a controller equivalent if there is a weak bisimulation between the branching process of the strategy and the plays that are compatible with the controller. We base our definition on a set of shared actions and transitions between the Petri game and the control game. We refer to them as *observable*. All non-shared transitions and actions are considered *internal* (τ). If we, e.g., translate a Petri game to a control game we aim for a control game that contains all transitions as observable actions but might add internal ones.

▶ **Definition 1.** *A strategy σ for $\mathcal{G}$ and controller ϱ for $\mathcal{C}$ are* bisimilar *if there exists a relation $\approx_\mathfrak{B} \subseteq \mathcal{R}(\mathcal{N}^\sigma) \times Plays(\mathcal{A}, \varrho)$ s.t. $In^\sigma \approx_\mathfrak{B} \epsilon$ and all following conditions hold:*

- *If $M \approx_\mathfrak{B} u$ and $M\,[\,a\,\rangle\,M'$ there exists $u' \in Plays(\mathcal{A}, \varrho)$ with $u' = u\tau^*a\tau^*$ and $M' \approx_\mathfrak{B} u'$*
- *If $M \approx_\mathfrak{B} u$ and $M\,[\,\tau\,\rangle\,M'$ there exists $u' \in Plays(\mathcal{A}, \varrho)$ with $u' = u\tau^*$ and $M' \approx_\mathfrak{B} u'$*
- *If $M \approx_\mathfrak{B} u$ and $u' = u\,a$ there exists $M' \in \mathcal{R}(\mathcal{N}^\sigma)$ with $M\,[\,\tau^*a\tau^*\,\rangle\,M'$ and $M' \approx_\mathfrak{B} u'$*
- *If $M \approx_\mathfrak{B} u$ and $u' = u\,\tau$ there exists $M' \in \mathcal{R}(\mathcal{N}^\sigma)$ with $M\,[\,\tau^*\,\rangle\,M'$ and $M' \approx_\mathfrak{B} u'$*

A Petri game $\mathcal{G}$ and a control game $\mathcal{C}$ are called *strategy-equivalent* if for every winning strategy σ for $\mathcal{G}$ there exists a bisimilar winning controller ϱ_σ for $\mathcal{C}$ and for every winning controller ϱ for $\mathcal{C}$ there exists a bisimilar winning strategy σ_ϱ for $\mathcal{G}$.

5 Translating Petri Games to Control Games

We give our translation from Petri games to control games and prove that it yields strategy-equivalent (and therefore winning-equivalent) games. Moreover, we provide an exponential lower bound, showing that our translation is asymptotically optimal when requiring strategy-equivalence. We present the translation for reachability objectives. Due to space restrictions, all proofs and further details can be found in the full version of the paper [3].

5.1 Construction

We describe the construction of our translation for a restrictive class of Petri games called *sliceable*. In Sec. 5.3, the construction is generalized to *concurrency-preserving* Petri games.

Slices. A Petri game describes the *global* behavior of the players. By contrast, a control game is defined in terms of *local* processes. Similarly, a Petri game strategy is a global branching process opposed to a family of local controllers for control games. The first difference our translation needs to overcome is to distribute a Petri game into parts describing the local behavior of players. Therefore, we dismantle the Petri game into *slices* for each token.

▶ **Definition 2.** *A slice of a Petri net $\mathcal{N}$ is a Petri net $\varsigma = (\mathcal{P}^\varsigma, \mathcal{T}^\varsigma, \mathcal{F}^\varsigma, In^\varsigma)$ s.t.,* **(1)** $\varsigma \sqsubseteq \mathcal{N}$, **(2)** $|In^\varsigma| = 1$, **(3)** $\forall t \in \mathcal{T}^\varsigma : |pre^\varsigma(t)| = |post^\varsigma(t)| = 1$, **(4)** $\forall q \in \mathcal{P}^\varsigma : post^{\mathcal{N}}(q) \subseteq \mathcal{T}^\varsigma$

A slice is a subnet of $\mathcal{N}$ **(1)** that describes the course of exactly one token **(2, 3)** and includes every possible move of this token **(4)**. A slice characterizes the exact behavior of a single token in the global net $\mathcal{N}$. For a family of slices $\{\varsigma\}_{\varsigma \in \mathbb{S}}$, the parallel composition $\|_{\varsigma \in \mathbb{S}}\ \varsigma$ is the Petri net with places $\biguplus_{\varsigma \in \mathbb{S}} \mathcal{P}^\varsigma$, transitions $\bigcup_{\varsigma \in \mathbb{S}} \mathcal{T}^\varsigma$, flow relation $\biguplus_{\varsigma \in \mathbb{S}} \mathcal{F}^\varsigma$, and initial marking $\biguplus_{\varsigma \in \mathbb{S}} In^\varsigma$. All unions, except for the union of transitions, are disjoint. Transitions can be shared between multiple slices, creating synchronization. A Petri net $\mathcal{N}$ is *sliceable* if there is a family of slices $\{\varsigma\}_{\varsigma \in \mathbb{S}}$ s.t. $\mathcal{N} = \|_{\varsigma \in \mathbb{S}}\ \varsigma$ and $\biguplus_{\varsigma \in \mathbb{S}} \mathcal{P}^\varsigma$ is a partition of $\mathcal{P}^{\mathcal{N}}$, i.e., $\mathcal{N}$ can be described by the local movements of tokens. Sliceable Petri nets are concurrency-preserving and 1-bounded. We extend slices to Petri games in the natural way by distinguishing system, environment, and special places. Figure 4 depicts a Petri game (a) and a possible distribution into slices (b). Note that even concurrency-preserving and 1-bounded Petri games must not be sliceable and that a distribution in slices is not unique.

Commitment Sets. In control games, actions are either controllable or uncontrollable whereas, in Petri games, players are distributed between the system and the environment. In our construction, we represent transitions as actions and need to guarantee that only certain players can control them. In control games, this cannot be expressed directly. We overcome this difference by using commitment sets. Each process that should be able to control an action chooses a commitment set, i.e., moves to a state that explicitly encodes its decision.

We fix a sliceable game $\mathcal{G} = (\mathcal{P}_\mathcal{S}, \mathcal{P}_\mathcal{E}, \mathcal{T}, \mathcal{F}, In, \mathcal{W})$ and a distribution in slices $\{\varsigma\}_{\varsigma \in \mathbb{S}}$. We begin by defining a control game $\mathcal{C}_\mathcal{G}$. Afterwards, we describe a possible modification $\widehat{\mathcal{C}_\mathcal{G}}$, that enforces determinism. The construction is depicted in Fig. 3.

Define $\mathfrak{P} = \mathfrak{S}$ and the distributed alphabet as (Σ, dom) with:

$$\Sigma = \mathcal{T} \;\cup\; \{\tau_{(q,A)} \mid q \in \mathcal{P}_{\mathcal{S}} \;\wedge\; A \subseteq post^{\mathcal{G}}(q)\}$$
$$\cup\; \{\, \lightning^{(q,A)}_{[t_1,t_2]} \mid q \in \mathcal{P}_{\mathcal{S}} \;\wedge\; A \subseteq post^{\mathcal{G}}(q) \;\wedge\; t_1, t_2 \in A \;\wedge\; t_1 \neq t_2\}$$

and $dom : \Sigma \to 2^{\mathfrak{P}} \setminus \{\emptyset\}$:

$$dom(\, t \,) = \{\varsigma \in \mathfrak{S} \mid t \in \mathcal{T}^{\varsigma}\} \quad \text{for } t \in \mathcal{T}$$
$$dom(\, \tau_{(q,A)} \,) = \{\varsigma\} \text{ where } \varsigma \in \mathfrak{S} \text{ is the unique slice s.t. } q \in \mathcal{P}^{\varsigma}$$
$$dom(\, \lightning^{(q,A)}_{[t_1,t_2]} \,) = \{\varsigma \in \mathfrak{S} \mid t_1 \in \mathcal{T}^{\varsigma} \;\vee\; t_2 \in \mathcal{T}^{\varsigma}\}$$

For each slice $\varsigma = (\mathcal{P}^{\varsigma}, \mathcal{T}^{\varsigma}, \mathcal{F}^{\varsigma}, In^{\varsigma}) \in \mathfrak{S}$, we define a local process $\mathfrak{A}_{\varsigma} = (Q_{\varsigma}, q_{0,\varsigma}, \vartheta_{\varsigma})$ with $\vartheta_{\varsigma} \subseteq Q_{\varsigma} \times \Sigma_{\varsigma} \times Q_{\varsigma}$ as:

- $Q_{\varsigma} = \mathcal{P}^{\varsigma} \cup \{(q, A) \mid q \in \mathcal{P}^{\varsigma} \cap \mathcal{P}_{\mathcal{S}} \;\wedge\; A \subseteq post^{\varsigma}(q)\} \cup \{\bot_{\varsigma}\}$
- $q_{0,\varsigma}$ is the unique state s.t. $In^{\varsigma} = \{q_{0,\varsigma}\}$

and ϑ_{ς} is given by:

(1) $q \xmapsto{\tau_{(q,A)}} (q, A)$

$q \in \mathcal{P}_{\mathcal{S}} \;\wedge$
$A \subseteq post^{\varsigma}(q)$

(2) $q \xmapsto{t} q'$

$q \in \mathcal{P}_{\mathcal{E}} \;\wedge\; t \in \mathcal{T} \;\wedge$
$q \in pre^{\varsigma}(t) \;\wedge\; q' \in post^{\varsigma}(t)$

(3) $(q, A) \xmapsto{t} q'$

$q \in \mathcal{P}_{\mathcal{S}} \;\wedge\; t \in A \;\wedge$
$q \in pre^{\varsigma}(t) \;\wedge\; q' \in post^{\varsigma}(t)$

(4) $(q, A) \xmapsto{\lightning^{(q,A)}_{[t_1,t_2]}} \bot_{\varsigma}$

(5) $q \xmapsto{\lightning^{(q',A')}_{[t_1,t_2]}} \bot_{\varsigma}$

$q' \notin Q_{\varsigma} \;\wedge\; q \in \mathcal{P}_{\mathcal{E}} \;\wedge$
$(t_1 \in \mathcal{T}^{\varsigma} \Rightarrow t_1 \in post^{\varsigma}(q)) \;\wedge$
$(t_2 \in \mathcal{T}^{\varsigma} \Rightarrow t_2 \in post^{\varsigma}(q))$

(6) $(q, A) \xmapsto{\lightning^{(q',A')}_{[t_1,t_2]}} \bot_{\varsigma}$

$q' \notin Q_{\varsigma} \;\wedge$
$(t_1 \in \mathcal{T}^{\varsigma} \Rightarrow t_1 \in A) \;\wedge$
$(t_2 \in \mathcal{T}^{\varsigma} \Rightarrow t_2 \in A)$

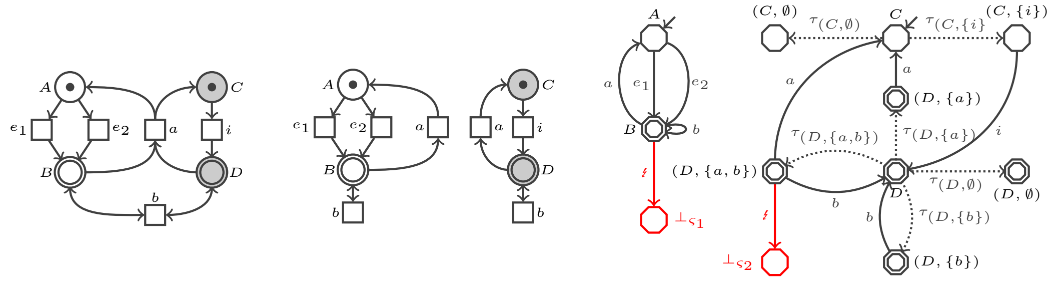
Define $\mathcal{A}_{\mathcal{G}} = \bigotimes_{\varsigma \in \mathfrak{S}} \mathfrak{A}_{\varsigma}$ and the control game as $\mathcal{C}_{\mathcal{G}} = (\mathcal{A}_{\mathcal{G}}, \Sigma^{sys}, \Sigma^{env}, \{\mathcal{W}_{\varsigma}\}_{\varsigma \in \mathfrak{S}})$ where

- $\Sigma^{sys} = \{\tau_{(q,A)} \mid q \in \mathcal{P}_{\mathcal{S}} \;\wedge\; A \subseteq post^{\mathcal{G}}(q)\}$
- $\Sigma^{env} = \mathcal{T} \;\cup\; \{\, \lightning^{(q,A)}_{[t_1,t_2]} \mid q \in \mathcal{P}_{\mathcal{S}} \;\wedge\; A \subseteq post^{\mathcal{G}}(q) \;\wedge\; t_1, t_2 \in A \;\wedge\; t_1 \neq t_2\}$
- $\mathcal{W}_{\varsigma} = (\mathcal{W} \;\cap\; \mathcal{P}^{\varsigma}) \;\cup\; \{(q, A) \mid q \in (\mathcal{W} \cap \mathcal{P}^{\varsigma}) \;\wedge\; A \subseteq post^{\mathcal{G}}(q)\}$

Figure 3 The construction of the translated control game for a Petri game $\mathcal{G} = (\mathcal{P}_{\mathcal{S}}, \mathcal{P}_{\mathcal{E}}, \mathcal{T}, \mathcal{F}, In, \mathcal{W})$, distributed in slices $\{\varsigma\}_{\varsigma \in \mathfrak{S}}$, is depicted. Excluding the red parts, this is the definition of $\mathcal{C}_{\mathcal{G}}$. Including the red parts, this is the definition of $\widehat{\mathcal{C}_{\mathcal{G}}}$.

We transform every slice ς into a process that is described by a local automaton $\mathfrak{A}_{\varsigma}$. Hence, we use the terms slice and process interchangeably. Every place in ς becomes a local state in $\mathfrak{A}_{\varsigma}$. The process starts in the state that corresponds to the initial place of the slice. For every *system* place q, we furthermore add the aforementioned commitment sets. These are states (q, A) representing every possible commitment, i.e., every $A \subseteq post^{\mathcal{G}}(q)$.

Every transition t is added as an uncontrollable action. Action t involves all processes with slices synchronizing on t. To choose a commitment set, we furthermore add controllable actions (τ-actions) that are local to each process. We assume that each process chooses at most one commitment set. The transition relation ϑ_{ς} is given by three rules: From every *system* place $q \in \mathcal{P}_{\mathcal{S}}$, a process can choose a commitment set using the corresponding τ-action **(1)**. From an environment place $q \in \mathcal{P}_{\mathcal{E}}$, t can fire if q is in the precondition of t ($q \in pre^{\varsigma}(t)$). The process is then moved to the state q' that corresponds to the place that is reached when



(a) **(b)** **(c)**

Figure 4 A sliceable Petri game $\mathcal{G}$ (a), a possible (in this case unique) distribution in slices $(\varsigma_1, \varsigma_2)$ (b), and the asynchronous automaton $\mathcal{C}_\mathcal{G}$ obtained by our translation (c). $\widehat{\mathcal{C}_\mathcal{G}}$ comprises additional $\bot$-states and one $\lightning^{(D,\{a,b\})}_{[a,b]}$-action (named $\lightning$) depicted in red.

firing t in the slice ($q' \in post^\varsigma(t)$) **(2)**. A process on an environment place can hence never restrict any actions; as in Petri games. For a system place, the rule is almost identical but only admits t if a commitment set has been chosen, that contains t **(3)**. Therefore, a process on a system place can control actions by choosing commitment sets; as in Petri games. States corresponding to winning places become winning states.

An example translation is depicted in Fig. 4. The Petri game (a) comprise two players starting in A and C. They can move to B and D using e_1, e_2, or i and afterwards synchronize on a or b. The Petri game can be distributed into slices (b). In the control game from our construction (c), the slice containing only environment places results in the local process on the left. For the system places in the other slice, commitment sets are added as states $\{C\} \times 2^{\{i\}}$ and $\{D\} \times 2^{\{a,b\}}$. The process can choose them using controllable τ-actions and the actions a, b, and i can only occur if included in the current set. The construction guarantees that only the second process can control transitions a and b, as in the Petri game.

Non-Determinism. In deterministic strategies, every system place allows transitions s.t. in every situation, there is at most one of them enabled. In $\mathcal{C}_\mathcal{G}$, the controller can choose arbitrary commitment sets and, thus, a winning controller can result in a *non-deterministic* strategy for $\mathcal{G}$. To ensure deterministic strategies, we want to penalize situations where a commitment set in $\mathcal{C}_\mathcal{G}$ is chosen s.t. two or more distinct actions from this set can be taken.

To achieve this, we define the modified game $\widehat{\mathcal{C}_\mathcal{G}}$. We equip each process with a $\bot$-state from which no winning configurations are reachable. Uncontrollable $\lightning$-actions move processes to $\bot$-states and thereby cause the system to lose. The situation to be covered comprises a process that has chosen a commitment set, i.e., is in a state (q, A), and two distinct actions t_1 and t_2 in A. For every such combination, we add a $\lightning^{(q,A)}_{[t_1,t_2]}$-action that involves all processes participating in t_1 or t_2 and can be taken exactly if (q, A) is a current state and both t_1 *and* t_2 could occur from the current global state. The three rules in ϑ add the $\lightning^{(q,A)}_{[t_1,t_2]}$-action to each process. It fires if one process is in state (q, A) **(4)** and all other involved processes are in states such that both $t_1, t_2 \in A$ are possible **(4, 5)**. To ensure that t_1 and t_2 can both be taken, we distinguish between system and environment places: Every process ς on an environment place needs to be in the right state, i.e., if ς is involved in t_i ($t_i \in \mathcal{T}^\varsigma$), then t_i is in the postcondition of its current place for $i = 1, 2$ **(5)**. If on a system place, t_i must not only be in the postcondition but also in the currently chosen commitment set **(6)**.

Size. In both $\mathcal{C}_\mathcal{G}$ and $\widehat{\mathcal{C}_\mathcal{G}}$, the size of the alphabet and number of local states is exponential in the number of transitions and linear in the number of places. For $\mathcal{C}_\mathcal{G}$, the blow-up in the alphabet can be kept polynomial by using a tree construction to choose commitment sets. For a bound on the number of outgoing transitions, both $\mathcal{C}_\mathcal{G}$ and $\widehat{\mathcal{C}_\mathcal{G}}$ are of polynomial size.

5.2 Correctness

We show that our translation yields strategy-equivalent games by outlining the translation of winning strategies and controllers between $\mathcal{G}$ and $\mathcal{C}_\mathcal{G}$. Both game types rely on causal information, i.e., a strategy/controller bases its decisions on every action/transition it took part in as well as all information it received upon communication. In $\mathcal{G}$, the information is carried by individual tokens. In our translation, we transform each slice for a token into a process that is involved in exactly the transitions that the slice it is build from takes part in, i.e., we preserve the communication architecture. At every point, all processes in $\mathcal{C}_\mathcal{G}$ possess the same information as their counterpart slices. Using the commitment set, our translation ensures that only processes on a state based on a system place can control any behavior. Therefore, a process and its counterpart slice have the same possibilities for control.

Translating a Strategy for $\mathcal{G}$ to a Controller for $\mathcal{C}_\mathcal{G}$. Given a winning strategy σ, we construct a controller ϱ_σ. The only states from which a process p can control any behavior (in terms of controllable actions) are of the form $q \in \mathcal{P}_\mathcal{S}$. σ decides for every system place which transitions to enable. Due to our construction, p can copy the decision of σ by choosing an appropriate commitment set. Therefore, ϱ_σ allows the same behavior as σ. If σ is deterministic then the commitments sets are chosen such that no $\lightning$-actions are possible.

Translating a Controller for $\mathcal{C}_\mathcal{G}$ to a Strategy for $\mathcal{G}$. Given a winning controller ϱ, we incrementally construct a strategy σ_ϱ. Every system place q in the partially constructed strategy can control which transitions are enabled. The place q belongs to some process. If on a state corresponding to a system place, this process can control all actions using its commitment sets. q enables exactly the transitions that the process has chosen as a commitment set. An environment place cannot control any behavior and neither can the process it belongs to. Hence, ϱ and σ_ϱ allow the same actions and transitions. A winning controller for $\widehat{\mathcal{C}_\mathcal{G}}$ additionally avoids any uncontrollable $\lightning$-actions and results in a deterministic strategy.

We obtain that $\mathcal{G}$ and $\mathcal{C}_\mathcal{G}$ are strategy-equivalent and that $\mathcal{G}$ and $\widehat{\mathcal{C}_\mathcal{G}}$ are strategy-equivalent if we require deterministic strategies for Petri games.

5.3 Generalization to Concurrency-Preserving Games

Our translation builds processes from a slice distribution of the Petri game. This limits the translation to sliceable games. The notion of slices is too strict: Our translation only requires to distribute the global behavior of the Petri game into local behavior, a *partitioning* of the places is not necessarily needed. We introduce the new concept of *singular nets* (SN). Similar to a slice, an SN describes the course of one token. Instead of being a subnet, it is equipped with a labeling function assigning to each node in the singular net a node in the original net. This labeling allows us to split up places and transitions by equally labelled copies enabling us to distribute every concurrency-preserving Petri net and game into singular nets. We can build our previous translation with an SN-distribution instead of a slice-distribution.

Define $\mathcal{G}_{\mathcal{C}} = (\mathcal{P}_{\mathcal{S}}, \mathcal{P}_{\mathcal{E}}, \mathcal{T}, \mathcal{F}, In, \mathcal{B})$ where

- $\mathcal{P}_{\mathcal{S}} = \bigcup_{p\in\mathcal{P}} S_p, \quad \mathcal{P}_{\mathcal{E}} = \{\, (s, A) \mid s \in \bigcup_{p\in\mathcal{P}} S_p,\ A \subseteq act(s) \cap \Sigma^{sys}\} \cup \{\bot^p_{DL} \mid p \in \mathfrak{P}\}$
- $\mathcal{T} = \{\, (a, B, \{A_s\}_{s\in B}) \mid a \in \Sigma^{env},\ B \in domain(\delta_a),\ A_s \subseteq act(s) \cap \Sigma^{sys}\} \cup$ **(1)**

  $\{\, (a, B, \{A_s\}_{s\in B}) \mid a \in \Sigma^{sys},\ B \in domain(\delta_a),\ A_s \subseteq act(s) \cap \Sigma^{sys},\ a \in A_s\} \cup$ **(2)**

  $\{\, \tau_{(s,A)} \mid s \in \bigcup_{p\in\mathcal{P}} S_p,\ A \subseteq act(s) \cap \Sigma^{sys}\} \cup \{t^M_{DL} \mid M \in \mathfrak{D}_{DL}\}$ **(3), (7)**
- $\mathcal{F} = \{\, \big((s, A)\,, (a, B, \{A_s\}_{s\in B}) \big) \mid s \in B,\ A_s = A\} \cup$ **(4)**

  $\{\, \big((a, B, \{A_s\}_{s\in B})\,, s' \big) \mid s' \in \delta_a(B)\} \cup$ **(5)**

  $\{\, \big(s, \tau_{(s,A)}\big)\} \cup \{\big(\tau_{(s,A)}, (s, A)\big)\,\} \cup$ **(6)**

  $\{\, \big(q, t^M_{DL}\big) \mid q \in M\} \cup \{\, \big(t^M_{DL}, \bot^p_{DL}\big) \mid p \in \mathfrak{P}\}$ **(8)**
- $In = s^{\mathcal{A}}_{in}$ and $\mathcal{B} = \bigcup_{p\in\mathcal{P}} \mathcal{B}_p \cup \bigcup_{p\in\mathfrak{P}} \bot^p_{DL}$

Figure 5 We give the construction of the translated Petri game $\mathcal{G}_{\mathcal{C}}$ for a control game $\mathcal{C} = (\mathcal{A}, \Sigma^{sys}, \Sigma^{env}, \{\mathcal{B}_p\}_{p\in\mathcal{P}})$ where $\mathcal{A} = (\{S_p\}_{p\in\mathcal{P}}, s^{\mathcal{A}}_{in}, \{\delta_a\}_{a\in\Sigma})$. The initial state $s^{\mathcal{A}}_{in}$ is viewed as a set. The gray parts penalize the artificial deadlocks, i.e., all markings in $\mathfrak{D}_{DL}$.

▶ **Theorem 3.** *For every concurrency-preserving Petri game $\mathcal{G}$, there exist control games $\mathcal{C}_{\mathcal{G}}$ and $\widehat{\mathcal{C}_{\mathcal{G}}}$ with an equal number of players such that* **(1)** *$\mathcal{G}$ and $\mathcal{C}_{\mathcal{G}}$ are strategy-equivalent and* **(2)** *$\mathcal{G}$ and $\widehat{\mathcal{C}_{\mathcal{G}}}$ are strategy-equivalent if we require deterministic Petri game strategies.*

5.4 Lower Bound

We can show that there is a family of Petri games such that every strategy-equivalent control game must have exponentially many local states. In a control game, either all or none of the players can restrict an action. By contrast, Petri games offer a finer granularity of control by allowing only some players to restrict a transition. The insight for the lower bound is to create a situation where a transition is shared between players but can only be controlled by one of them. Using careful reasoning, we can show that in any strategy-equivalent control game there must be actions that can only be controlled by a single process, resulting in exponentially many local states. Our translation shows that the difference between both formalism can be overcome but our lower bound shows an intrinsic difficulty to achieve this.

▶ **Theorem 4.** *There is a family of Petri games such that every strategy-equivalent control game (with an equal number of players) must have at least $\Omega(d^n)$ local states for $d > 1$.*

6 Translating Control Games to Petri Games

We give our translation from control games to Petri games, prove that it yields strategy-equivalent games, and give an exponential lower bound. We present our translation for safety objectives. All proofs and further details can be found in the full version of this paper [3].

6.1 Construction

We fix a control game $\mathcal{C} = (\mathcal{A}, \Sigma^{sys}, \Sigma^{env}, \{\mathcal{B}_p\}_{p\in\mathcal{P}})$ with safety objective. The translation to $\mathcal{G}_{\mathcal{C}}$ is depicted in Fig. 5. We represent each local state s as a system place. We add environment places (s, A), which encode every possible commitment set of actions that can be allowed by a controller ($A \subseteq act(s) \cap \Sigma^{sys}$). From each system place, the player can move to

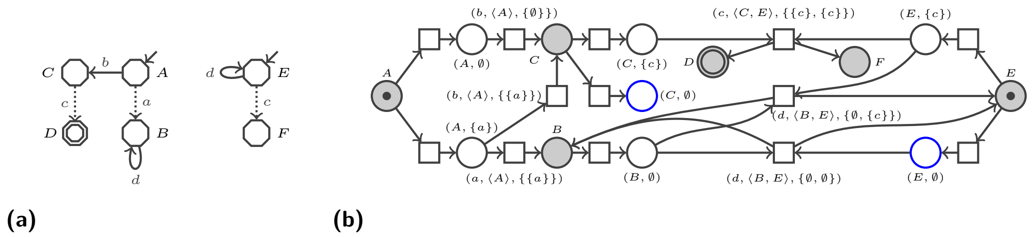

C
b
A
c
a
D
B
d
d
E
c
F
(a)
$(b, \langle A \rangle, \{\emptyset\}\})$
$(A, \emptyset)$
C
$(c, \langle C, E \rangle, \{\{c\}, \{c\}\})$
$(E, \{c\})$
$(C, \{c\})$
D
F
A
E
$(b, \langle A \rangle, \{\{a\}\})$
$(C, \emptyset)$
$(d, \langle B, E \rangle, \{\emptyset, \{c\}\})$
$(A, \{a\})$
B
$(a, \langle A \rangle, \{\{a\}\})$
$(B, \emptyset)$
$(d, \langle B, E \rangle, \{\emptyset, \emptyset\})$
$(E, \emptyset)$
(b)

Figure 6 Control game $\mathcal{C}$ (a) and translated Petri game $\mathcal{G}_{\mathcal{C}}$ (b) are given. Commitment sets without outgoing transitions are omitted. The set of artificial deadlocks $\mathfrak{D}_{DL}$ comprises every final marking that contains at least one blue place. The resulting t_{DL}^{M}-transitions are omitted.

places for the commitment sets using a $\tau_{(s,A)}$-transition **(3, 6)**. Each action a in $\mathcal{C}$ can occur from different configurations of the processes in $dom(a)$, i.e., all states in $domain(\delta_a)$, whereas in Petri games transitions fire from fixed preconditions. We want to represent a as a transition that fires from places representing commitment sets that correspond to configurations from which a can occur in $\mathcal{C}$. We hence duplicate a into multiple transitions to account for every configuration in $domain(\delta_a)$ and for every combination of commitment sets. Transitions have the form $(a, B, \{A_s\}_{s \in B})$ where a is the action in the control game, $B \in domain(\delta_a)$ is the configuration from which a can fire, and $\{A_s\}_{s \in B}$ are the involved commitment sets. If action a is uncontrollable the corresponding transitions are added independently of the commitment sets **(1)**. If a is controllable a transition is only added if a is in the commitment sets of all involved players, i.e., $a \in A_s$ for every $s \in B$ **(2)**. If $(a, B, \{A_s\}_{s \in B})$ is added it fires from precisely the precondition that is encoded in it, i.e., the places (s, A) where $s \in B$ and $A_s = A$, and moves every token to the system place that corresponds to the resulting local state when firing a in $\mathcal{C}$ **(4, 5)**. A strategy can restrict controllable actions by moving to an appropriate commitment set but cannot forbid uncontrollable ones, since they can occur from every combination of commitment sets. If a system player decides to refuse any commitment set it could prohibit transitions that correspond to uncontrollable actions. In Sec. 6.3, we show how to force the system to always choose a commitment set.

In safety games, every winning strategy must avoid deadlocks. By introducing explicit commitment sets, we add *artificial* deadlocks, i.e., configurations that are deadlocked in $\mathcal{G}_{\mathcal{C}}$ but where the corresponding state in $\mathcal{C}$ could still act. This permits trivial strategies that, e.g., always choose the empty commitment set. We define $\mathfrak{D}_{DL}$ as the set of all reachable markings that are final in $\mathcal{G}_{\mathcal{C}}$ but where the corresponding global state in $\mathcal{C}$ can still perform an action, i.e., all artificial deadlocks. Similar to the $\not{\iota}$-actions, we introduce t_{DL}^{M}-transitions that fire from every marking M in $\mathfrak{D}_{DL}$ and move every token to a losing place $\bot_{DL}$ **(7, 8)**. The mechanism to detect artificial deadlocks is depicted as the gray parts in Fig. 5.

Figure 6 depicts an example translation. The system cannot win this game: The uncontrollable action b can always happen, independent of the commitment set for place A. If one of the two tokens refuses c (moves to a blue place) a (losing) transition t_{DL} can fire.

6.2 Correctness

We show strategy-equivalence of $\mathcal{C}$ and $\mathcal{G}_{\mathcal{C}}$ by translating strategies (that always commit) and controllers between both of them. We observe that each token moves on the local states of one process and takes part in precisely the actions of the process. At every point, a token hence possesses the same local information as the process. A token can restrict the controllable actions using the commitment sets but cannot restrict the uncontrollable ones. The token therefore has the same possibilities as the process counterpart.

Translating Controllers to Strategies. Given a winning controller ϱ, we incrementally build a (possibly infinite) winning, deterministic strategy σ_ϱ. Every system place q in a partially constructed strategy can choose one of the commitment sets. q copies ϱ by committing to exactly the actions that the process it belongs to has allowed. The commitment sets can only restrict controllable actions, as the process can. Hence, σ_ϱ allows the same behavior as ϱ.

Translating Strategies to Controllers. Given a winning, deterministic strategy σ, we construct a winning controller ϱ_σ. A process p that resides on a local state s can decide which of the controllable actions should be allowed. Every token in σ can decide for a commitment set and therefore implicitly chooses which controllable actions should be enabled. p allows exactly the actions that σ chooses as a commitment set. Both can only restrict controllable actions and, by copying, ϱ_σ achieves the same behavior as σ.

▶ **Theorem 5.** *$\mathcal{C}$ and $\mathcal{G}_\mathcal{C}$ are strategy-equivalent.*

6.3 Enforcing Commitment

Our construction assumes wining strategies to always choose a commitment set. We can modify $\mathcal{G}_\mathcal{C}$ such that every non-committing strategy cannot win. The insight is to use the deadlock-avoidance of winning strategies. Deadlocks define a *global* situation of the game. To enforce commitment, we require *local* deadlock-avoidance in the sense that every token has to choose a commitment set. This is not prevented by global deadlock-avoidance, where, e.g., a single player being able to play locally enables every other player to refuse to commit without being deadlocked. We reduce local to global deadlocks by adding transitions to challenge the players to have reached a local deadlock. Using challenge transitions, every player currently residing on a place that corresponds to a chosen commitment set moves to a terminating place. Every player that has chosen commitment sets can terminate, resulting in the players that are locally deadlocked to cause a global deadlock. Although the challenge is always possible, the scheduler decides the point of challenge. The game with the added challenger has a winning strategy iff $\mathcal{G}_\mathcal{C}$ has a winning strategy that always commits.

6.4 Lower Bounds

We can provide a family of control games where every strategy-equivalent Petri game must be of exponential size. In control games, both controllable and uncontrollable actions can occur from the same state. In Petri games, a given place can either restrict all transitions (system place) or none. A control game where both actions types are possible already results in Petri games of exponential size. We assume the absence of infinite τ-sequences.

▶ **Theorem 6.** *There is a family of control games such that every strategy-equivalent Petri game (with an equal number of players) must have at least $\Omega(d^n)$ places for $d > 1$.*

7 New Decidable Classes

We exemplarily show one transferrable class of decidability for both control games and Petri games to highlight the applicability of our translations.

New Decidable Control Games. A process in a control game is an *environment process* if all its action are uncontrollable. A *system process* is one that is not an environment process. We can modify our second translation by not adding system places if there are no outgoing controllable actions. Therefore, environment processes do not add system places to the Petri game and we can use the results from [10].

▶ **Corollary 7.** *Control games with safety objectives and one system process are decidable.*

New Decidable Petri Games. Given a Petri game $\mathcal{G}$ and a distribution into slices (or SNs) $\{\varsigma\}_{\varsigma \in \mathcal{S}}$, we analyze the communication structure between the slices by building the undirected graph (V, E) where $V = \mathcal{S}$ and $E = \{(\varsigma_1, \varsigma_2) \mid \mathcal{T}^{\varsigma_1} \cap \mathcal{T}^{\varsigma_2} \neq \emptyset\}$. (V, E) is isomorphic to the *communication architecture* of the constructed asynchronous automaton $\mathcal{C}_{\mathcal{G}}$ (as introduced in [13]). We define $\mathcal{G}_{\circlearrowleft}$ as every Petri game that has a distribution $\{\varsigma\}_{\varsigma \in \mathcal{S}}$ where (V, E) is *acyclic*. We can show that such distributions are hard to find. From [13], we obtain decidability.

▶ **Lemma 8.** *Deciding whether a Petri net has an acyclic slice-distribution is* NP*-complete.*

▶ **Corollary 9.** *Petri games in* $\mathcal{G}_{\circlearrowleft}$ *with reachability objectives are decidable.*

8 Conclusion

We have provided the first formal connection between control games and Petri games by showing that both are equivalent. This indicates that synthesis models for asynchronous systems with causal memory are stable under the concrete formalisms of system and environment responsibilities for the two most common models. Conversely, our lower bounds show an intrinsic difference between control games and Petri games. By our translations, existing and future decidability results can be combined and transferred between both game types. Our translations could be adapted to other winning objectives. An interesting direction for future work is to investigate how action-based control games [21] relate to Petri games and to study unified models that combine features from control games and Petri games.

References

1 Cyril Autant and Philippe Schnoebelen. Place Bisimulations in Petri Nets. In *Proceedings of Application and Theory of Petri Nets*, pages 45–61, 1992. doi:10.1007/3-540-55676-1_3.

2 Eike Best, Raymond R. Devillers, Astrid Kiehn, and Lucia Pomello. Concurrent Bisimulations in Petri Nets. *Acta Inf.*, 28(3):231–264, 1991. doi:10.1007/BF01178506.

3 Raven Beutner, Bernd Finkbeiner, and Jesko Hecking-Harbusch. Translating Asynchronous Games for Distributed Synthesis (Full Version). *arXiv preprint*, 2019. arXiv:1907.00829.

4 J. Richard Buchi and Lawrence H. Landweber. Solving sequential conditions by finite state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

5 Joost Engelfriet. Branching Processes of Petri Nets. *Acta Inf.*, 28(6):575–591, 1991. doi:10.1007/BF01463946.

6 Javier Esparza and Keijo Heljanko. *Unfoldings – A Partial-Order Approach to Model Checking.* Springer, 2008. doi:10.1007/978-3-540-77426-6.

7 Bernd Finkbeiner. Bounded Synthesis for Petri Games. In *Proceedings of Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday*, pages 223–237, 2015. doi:10.1007/978-3-319-23506-6_15.

8 Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Symbolic vs. Bounded Synthesis for Petri Games. In *Proceedings of SYNT@CAV*, pages 23–43, 2017. doi:10.4204/EPTCS.260.5.

9 Bernd Finkbeiner, Manuel Gieseking, and Ernst-Rüdiger Olderog. Adam: Causality-Based Synthesis of Distributed Systems. In *Proceedings of CAV*, pages 433–439, 2015. doi:10.1007/978-3-319-21690-4_25.

10 Bernd Finkbeiner and Paul Gölz. Synthesis in Distributed Environments. In *Proceedings of FSTTCS*, pages 28:1–28:14, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.28.

11 Bernd Finkbeiner and Ernst-Rüdiger Olderog. Petri games: Synthesis of distributed systems with causal memory. *Inf. Comput.*, 253:181–203, 2017. doi:10.1016/j.ic.2016.07.006.

12 Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed Games with Causal Memory Are Decidable for Series-Parallel Systems. In *Proceedings of FSTTCS*, pages 275–286, 2004. doi:10.1007/978-3-540-30538-5_23.

13 Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Asynchronous Games over Tree Architectures. In *Proceedings of ICALP*, pages 275–286, 2013. doi:10.1007/978-3-642-39212-2_26.

14 Hugo Gimbert. On the Control of Asynchronous Automata. In *Proceedings of FSTTCS*, pages 30:1–30:15, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.30.

15 Jesko Hecking-Harbusch and Niklas O. Metzger. Efficient Trace Encodings of Bounded Synthesis for Asynchronous Distributed Systems. In *Proceedings of ATVA*, 2019.

16 P. Madhusudan and P. S. Thiagarajan. A Decidable Class of Asynchronous Distributed Controllers. In *Proceedings of CONCUR*, pages 145–160, 2002. doi:10.1007/3-540-45694-5_11.

17 P. Madhusudan, P. S. Thiagarajan, and Shaofa Yang. The MSO Theory of Connectedly Communicating Processes. In *Proceedings of FSTTCS*, pages 201–212, 2005. doi:10.1007/11590156_16.

18 José Meseguer, Ugo Montanari, and Vladimiro Sassone. Process versus Unfolding Semantics for Place/Transition Petri Nets. *Theor. Comput. Sci.*, 153(1&2):171–210, 1996. doi:10.1016/0304-3975(95)00121-2.

19 Anca Muscholl. Automated Synthesis of Distributed Controllers. In *Proceedings of ICALP*, pages 11–27, 2015. doi:10.1007/978-3-662-47666-6_2.

20 Anca Muscholl and Igor Walukiewicz. Distributed Synthesis for Acyclic Architectures. In *Proceedings of FSTTCS*, pages 639–651, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.639.

21 Anca Muscholl, Igor Walukiewicz, and Marc Zeitoun. A look at the control of asynchronous automata. *Perspectives in Concurrency Theory*, pages 356–371, 2009.

22 Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri Nets, Event Structures and Domains, Part I. *Theor. Comput. Sci.*, 13:85–108, 1981. doi:10.1016/0304-3975(81)90112-2.

23 Ernst-Rüdiger Olderog. *Nets, terms and formulas: three views of concurrent processes and their relationship*, volume 23. Cambridge University Press, 2005.

24 Amir Pnueli and Roni Rosner. Distributed Reactive Systems Are Hard to Synthesize. In *31st Annual Symposium on Foundations of Computer Science, 1990, Volume II*, pages 746–757, 1990. doi:10.1109/FSCS.1990.89597.

25 Wolfgang Reisig. *Petri Nets: An Introduction.* Springer, 1985. doi:10.1007/978-3-642-69968-9.

26 Wieslaw Zielonka. Notes on Finite Asynchronous Automata. *ITA*, 21(2):99–135, 1987. doi:10.1051/ita/1987210200991.

Long-Run Average Behavior of Vector Addition Systems with States

Krishnendu Chatterjee
IST Austria, Klosterneuburg, Austria
krish.chat@ist.ac.at

Thomas A. Henzinger
IST Austria, Klosterneuburg, Austria
tah@ist.ac.at

Jan Otop
University of Wrocław, Poland
jotop@cs.uni.wroc.pl

Abstract

A vector addition system with states (VASS) consists of a finite set of states and counters. A configuration is a state and a value for each counter; a transition changes the state and each counter is incremented, decremented, or left unchanged. While qualitative properties such as state and configuration reachability have been studied for VASS, we consider the long-run average cost of infinite computations of VASS. The cost of a configuration is for each state, a linear combination of the counter values. In the special case of uniform cost functions, the linear combination is the same for all states. The (regular) long-run emptiness problem is, given a VASS, a cost function, and a threshold value, if there is a (lasso-shaped) computation such that the long-run average value of the cost function does not exceed the threshold. For uniform cost functions, we show that the regular long-run emptiness problem is (a) decidable in polynomial time for integer-valued VASS, and (b) decidable but nonelementarily hard for natural-valued VASS (i.e., nonnegative counters). For general cost functions, we show that the problem is (c) NP-complete for integer-valued VASS, and (d) undecidable for natural-valued VASS. Our most interesting result is for (c) integer-valued VASS with general cost functions, where we establish a connection between the regular long-run emptiness problem and quadratic Diophantine inequalities. The general (nonregular) long-run emptiness problem is equally hard as the regular problem in all cases except (c), where it remains open.

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

Keywords and phrases vector addition systems, mean-payoff, Diophantine inequalities

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.27

Related Version A full version of the paper is available at `http://arxiv.org/abs/1905.05537`.

Funding *Krishnendu Chatterjee*: The Austrian Science Fund (FWF) NFN grant S11407-N23 (RiSE/SHiNE)
Thomas A. Henzinger: The Austrian Science Fund (FWF) grants S11402-N23 (RiSE/ShiNE) and Z211-N23 (Wittgenstein Award)
Jan Otop: The National Science Centre (NCN), Poland under grant 2017/27/B/ST6/00299

Acknowledgements We thank Petr Novotný for helpful discussion on the literature.

1 Introduction

Vector Addition System with States (VASS). *Vector Addition Systems (VASS)* [20] are equivalent to Petri Nets, and they provide a fundamental framework for formal analysis of parallel processes [14]. The extension of VASs with a finite-state transition structure

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 27; pp. 27:1–27:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

gives *Vector Addition Systems with States (VASS)*. Informally, a VASS consists of a finite set of control states and transitions between them, and a set of k counters, where at every transition between the control states each counter is either incremented, decremented, or remains unchanged. A *configuration* consists of a control state and a valuation of each counter, and the transitions of the VASS describe the transitions between the configurations. Thus a VASS represents a finite description of an infinite-state transition system between the configurations. If the counters can hold all possible integer values, then we call them integer-valued VASS; and if the counters can hold only non-negative values, then we call them natural-valued VASS.

Modelling power. VASS are a fundamental model for concurrent processes [14], and often used in performance analysis of concurrent processes [12, 16, 18, 19]. Moreover, VASS have been used (a) in analysis of parametrized systems [2], (b) as abstract models for programs for bounds and amortized analysis [29], (c) in interactions between components of an API in component-based synthesis [15]. Thus they provide a rich framework for a variety of problems in verification as well as program analysis.

Previous results for VASS. A computation (or a run) in a VASS is a sequence of configurations. The well-studied problems for VASS are as follows: (a) *control-state reachability* where given a set of target control states a computation is successful if one of the target states is reached; (b) *configuration reachability* where given a set of target configurations a computation is successful if one of the target configurations is reached. For natural-valued VASS, (a) the control-state reachability problem is ExpSpace-complete: the ExpSpace-hardness is shown in [25, 13] and the upper bound follows from [28]; and (b) the configuration reachability problem is decidable [26, 21, 22, 23], and a recent breakthrough result shows that the problem is non-elementary hard [10]. For integer-valued VASS, (a) the control-state reachability problem is NLogSpace-complete: the counters can be abstracted away and we have to solve the reachability problem for graphs; and (b) the configuration reachability problem is NP-complete: this is a folklore result obtained via reduction to linear Diophantine inequalities.

Long-run average property. The classical problems for VASS are qualitative (or Boolean) properties where each computation is either successful or not. In this work we consider long-run average property that assigns a real value to each computation. A cost is associated to each configuration and the value of a computation is the long-run average of the costs of the configurations of the computation. For cost assignment to configuration we consider linear combination with natural coefficients of the values of the counters. In general the linear combination for the cost depends on the control state of the configuration, and in the special case of uniform cost functions the linear combination is the same across all states.

Motivating examples. We present some motivating examples for the problems we consider. First, consider a VASS where the counters represent different queue lengths, and each queue consumes resource (e.g., energy) proportional to its length, however, for different queues the constant of the proportionality might differ. The computation of the long-run average resource consumption is modeled as a uniform cost function, where the linear combination for the cost function is obtained from the constants of proportionality. Second, consider a system that uses two different batteries, and the counters represent the charge levels. At different states, different batteries are used, and we are interested in the long-run average charge of the used battery. This is modeled as a general cost function, where depending on the control state the cost function is the value of the counter representing the battery used in the state.

Our contributions. We consider the following decision problem: given a VASS and a cost function decide whether there is a regular (or periodic) computation such that the long-run average value is at most a given threshold. Our main contributions are as follows:

1. For uniform cost functions, we show that the problem is (a) decidable in polynomial time for integer-valued VASS, and (b) decidable but non-elementary hard for natural-valued VASS. In (b) we assume that the cost function depends on all counters.
2. For general cost functions, we show that the problem is (a) NP-complete for integer-valued VASS, and (b) undecidable for natural-valued VASS.

Our most interesting result is for general cost functions and integer-valued VASS, where we establish an interesting connection between the problem we consider and quadratic Diophantine inequalities. Finally, instead of regular computations, if we consider existence of an arbitrary computation, then all the above results hold, other than the NP-completeness result for general cost functions and integer-valued VASS (which remains open).

Related works. Long-run average behavior have been considered for probabilistic VASS [5], and other infinite-state models such as pushdown automata and games [9, 8, 1]. In these works the costs are associated with transitions of the underlying finite-state representation. In contrast, in our work the costs depend on the counter values and thus on the configurations. Costs based on configurations, specifically the content of the stack in pushdown automata, have been considered in [27]. Quantitative asymptotic bounds for polynomial-time termination in VASS have also been studied [4, 24], however, these works do not consider long-run average property. Finally, a related model of automata with monitor counters with long-run average property have been considered in [7, 6]. However, there are some crucial differences: in automata with monitor counters, the cost always depends on one counter, and counters are reset once the value is used. Moreover, the complexity results for automata with monitor counters are quite different from the results we establish.

2 Preliminaries

For a sequence w, we define $w[i]$ as the $(i+1)$-th element of w (we start with 0) and $w[i,j]$ as the subsequence $w[i]w[i+1]\ldots w[j]$. We allow j to be ∞ for infinite sequences. For a finite sequence w, we denote by $|w|$ its length; and for an infinite sequence the length is ∞.

We use the same notation for vectors. For a vector $\vec{x} \in \mathbb{R}^k$ (resp., $\mathbb{Q}^k$, $\mathbb{Z}^k$ or $\mathbb{N}^k$), we define $x[i]$ as the i-th component of $\vec{x}$. We define the support of $\vec{x}$, denoted by $supp(\vec{x})$ as the set of components of $\vec{x}$ with non-zero values. For vectors $\vec{x}, \vec{y}$ of equal dimension, we denote by $\vec{x} \cdot \vec{y}$, the dot-product of $\vec{x}$ and $\vec{y}$.

2.1 Vector addition systems with states (VASS)

A k-dimensional vector addition system with states (VASS) over $\mathbb{Z}$ (resp., over $\mathbb{N}$), referred to as $\text{VASS}(\mathbb{Z}, k)$ (resp., $\text{VASS}(\mathbb{N}, k)$), is a tuple $\mathcal{A} = \langle Q, Q_0, \delta \rangle$, where (1) Q is a finite set of states, (2) $Q_0 \subseteq Q$ is a set of initial states, and (3) $\delta \subseteq Q \times Q \times \mathbb{Z}^k$. We denote by $\mathbf{VASS}(\mathbb{Z}, k)$ (resp., $\mathbf{VASS}(\mathbb{N}, k)$) the class of k-dimensional VASS over $\mathbb{Z}$ (resp., $\mathbb{N}$). We often omit the dimension in VASS and write $\text{VASS}(\mathbb{Z}), \text{VASS}(\mathbb{N}), \mathbf{VASS}(\mathbb{N}), \mathbf{VASS}(\mathbb{Z})$ if a definition or an argument is uniform w.r.t. the dimension.

Configurations and computations. A *configuration* of a $\mathrm{VASS}(\mathbb{Z}, k)$ $\mathcal{A}$ is a pair from $Q \times \mathbb{Z}^k$, which consists of a state and a valuation of the counters. A *computation* of $\mathcal{A}$ is an infinite sequence π of configurations such that (a) $\pi[0] \in Q_0 \times \{\vec{0}\}$[1], and (b) for every $i \geq 0$, there exists $(q, q', \vec{y}) \in \delta$ such that $\pi[i] = (q, \vec{x})$ and $\pi[i+1] = (q', \vec{x} + \vec{y})$. A computation of a $\mathrm{VASS}(\mathbb{N}, k)$ $\mathcal{A}$ is a computation π of $\mathcal{A}$ considered as a $\mathrm{VASS}(\mathbb{Z}, k)$ such that the values of all counters are natural, i.e., for all $i > 0$ we have $\pi[i] \in Q \times \mathbb{N}^k$.

Paths and cycles. A path $\rho = (q_0, q_0', \vec{y}_0), (q_1, q_1', \vec{y}_1), \ldots$ in a $\mathrm{VASS}(\mathbb{Z})$ (resp., $\mathrm{VASS}(\mathbb{N})$) $\mathcal{A}$ is a (finite or infinite) sequence of transitions (from δ) such that for all $0 \leq i \leq |\rho|$ we have $q_i' = q_{i+1}$. A finite path ρ is a cycle if $\rho = (q_0, q_0', \vec{y}_0), \ldots, (q_m, q_m', \vec{y}_m)$ and $q_0 = q_m'$. Every computation in a $\mathrm{VASS}(\mathbb{Z})$ (resp., $\mathrm{VASS}(\mathbb{N})$) defines a unique infinite path. Conversely, every infinite path in a $\mathrm{VASS}(\mathbb{Z})$ $\mathcal{A}$ starting with $q_0 \in Q_0$ defines a computation in $\mathcal{A}$. However, if $\mathcal{A}$ is a $\mathrm{VASS}(\mathbb{N}, k)$, some paths do not have corresponding computations due to non-negativity restriction posed on the counters.

Regular computations. We say that a computation π of a $\mathrm{VASS}(\mathbb{Z})$ (resp., $\mathrm{VASS}(\mathbb{N})$) is *regular* if it corresponds to a path which is ultimately periodic, i.e., it is of the form $\alpha\beta^\omega$ where α, β are finite paths.

2.2 Decision problems

We present the decision problems that we study in the paper.

Cost functions. Consider a $\mathrm{VASS}(\mathbb{Z}, k)$ (resp., $\mathrm{VASS}(\mathbb{N}, k)$) $\mathcal{A} = (Q, Q_0, \delta)$. A cost function f for $\mathcal{A}$ is a function $f \colon Q \times \mathbb{Z}^k \to \mathbb{Z}$, which is linear with natural coefficients, i.e., for every q there exists $\vec{a} \in \mathbb{N}^k$ such that $f(q, \vec{z}) = \vec{a} \cdot \vec{z}$ for all $\vec{z} \in \mathbb{Z}^k$. We extend cost functions to computations as follows. For a computation π of $\mathcal{A}$, we define $f(\pi)$ as the sequence $f(\pi[0]), f(\pi[1]), \ldots$. Every cost function f is given by a labeling $l \colon Q \to \mathbb{N}^k$ by $f(q, \vec{z}) = l(q) \cdot \vec{z}$. We define the size of f, denoted by $|f|$, as the size of binary representation of l considered as a sequence of natural numbers of the length $|Q| \cdot k$.

Uniform cost functions. We say that a cost function f is *uniform*, if it is given by a constant function l, i.e., for all states q, q' and $\vec{z} \in \mathbb{Z}^k$ we have $f(q, \vec{z}) = f(q', \vec{z})$. Uniform cost functions are given by a single vector $\vec{a} \in \mathbb{N}^k$.

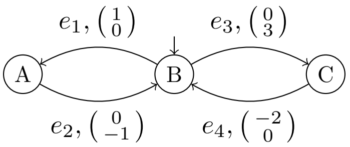
The long-run average. We are interested in the long-run average of the values returned by the cost function, which is formalized as follows. Consider an infinite sequence of real numbers w. We define $\textsc{LimAvg}(w) = \liminf_{k \to \infty} \frac{1}{k+1} \sum_{i=0}^{k} w[i]$.

▶ **Definition 1** (The average-value problems). *Given a* $\mathrm{VASS}(\mathbb{Z}, k)$ *(resp.,* $\mathrm{VASS}(\mathbb{N}, k)$*)* $\mathcal{A}$*, a cost function* f *for* $\mathcal{A}$*, and a threshold* $\lambda \in \mathbb{Q}$*,*

- *the* ***average-value*** *problem (resp., the* ***regular average-value*** *problem) asks whether* $\mathcal{A}$ *has a computation (resp., a regular computation)* π *such that* $\textsc{LimAvg}(f(\pi)) \leq \lambda$*, and*
- *the* ***finite-value*** *problem (resp., the* ***regular finite-value*** *problem) asks whether* $\mathcal{A}$ *has a computation (resp., a regular computation)* π *such that* $\textsc{LimAvg}(f(\pi)) < \infty$*.*

The following example illustrates the regular average-value problem for $\mathbf{VASS}(\mathbb{Z})$.

[1] Without loss of generality, we assume that the initial counter valuation is $\vec{0}$. We can encode any initial configuration in the VASS itself.



$$A \mapsto \binom{4}{0}$$
$$B \mapsto \binom{1}{1}$$
$$C \mapsto \binom{0}{1}$$

Figure 1 The VASS $\mathcal{A}_e$ and its labeling.

▶ **Example 2.** Consider the VASS$(\mathbb{Z}, 2)$ $\mathcal{A}_e = \langle \{A, B, C\}, \{B\}, \{e_1, e_2, e_3, e_4\}\rangle$ depicted in Figure 1 and a (non-uniform) cost function f is given by the labeling from Figure 1. $A \mapsto \binom{4}{0}$,

Consider an infinite path $(e_1e_2e_3e_4)^\omega$ that defines the regular computation π_0. This computation can be divided into blocks of 4 consecutive configurations. The $(i+1)$-th block has the following form

$$\dots (B, \binom{-i}{2i})(A, \binom{-i+1}{2i})(B, \binom{-i+1}{2i-1})(C, \binom{-i+1}{2i+2}) \dots$$

and the corresponding values of $f(\pi)$ are:

$$\dots \quad i \quad -4i+4 \quad i \quad 2i+2 \quad \dots$$

Therefore, the sum of values over each block is 6 and hence $\textsc{LimAvg}(f(\pi_0)) = \frac{3}{2}$.

The answer to the (regular) average-value problem with any threshold is YES. Consider $j \in \mathbb{N}$ and a path $(e_1e_2)^j(e_1e_2e_3e_4)^\omega$. Observe that it defines a regular computation π_j, which after the block $(e_1e_2)^j$ coincides with π_0 with all counter values shifted by $\binom{-2j}{3j}$. Note that in each block $e_1e_2e_3e_4$, the value f on the vector $\binom{-2j}{3j}$ are $j, -8j, j, 3j$. Therefore, $\textsc{LimAvg}(f(\pi_j)) = \textsc{LimAvg}(f(\pi_0)) + \frac{-3j}{4} = \frac{-3j-6}{4}$. It follows that for every threshold $\lambda \in \mathbb{Q}$, there exists a regular computation π_j with $\textsc{LimAvg}(f(\pi_j)) \leq \lambda$.

Organization. In this paper, we study the (regular) average-value problem for $\mathbf{VASS}(\mathbb{Z}, k)$ and $\mathbf{VASS}(\mathbb{N}, k)$. First, we study the average-value problem for uniform cost functions (Section 3). Next, we consider the non-uniform case, where we focus on $\mathbf{VASS}(\mathbb{Z})$ (Section 4). We start with solving the (regular) finite-value problem, and then we move to the regular average-value problem. We show that both problems are NP-complete. Next, we discuss the non-uniform case for $\mathbf{VASS}(\mathbb{N})$ and we show that the regular finite-value problem is decidable (and non-elementary), while the (regular) average-value problem is undecidable.

3 Uniform cost functions

In this section we study the average-value problem for $\mathbf{VASS}(\mathbb{Z})$ and $\mathbf{VASS}(\mathbb{N})$ for the uniform cost functions.

3.1 Integer-valued VASS: VASS$(\mathbb{Z})$

Consider a VASS$(\mathbb{Z}, k)$ $\mathcal{A}$ and a uniform cost function f for $\mathcal{A}$. This function is defined by a single vector of coefficients $\vec{a} \in \mathbb{N}^k$. First, observe that we can reduce the number of the counters to one, which tracks the current cost. This counter c stores the value of $f(\pi[i]) = \vec{a} \cdot \vec{z}$, where $\vec{z}$ are the values of counters. Initially, the counter c is $0 = \vec{a} \cdot \vec{0}$. Next, in each step $i > 0$, counters $\vec{z}$ are updated with some values $\vec{y}$ and we update c with $\vec{a} \cdot \vec{y}$. Note that the value of $f(\pi[i+1]) = \vec{a} \cdot (\vec{z} + \vec{y})$ equals $f(\pi[i]) + \vec{a} \cdot \vec{y}$.

Second, observe that the average-value problem for $\mathbf{VASS}(\mathbb{Z}, 1)$ and uniform cost functions is equivalent to single-player average energy games (with no bounds on energy levels), which are solvable in polynomial time [3]. Moreover, average-energy games (with no bounds on energy levels) admit memoryless winning strategies and hence the average-value and the regular average-value problems coincide. In consequence we have:

▶ **Theorem 3.** *The average-value and the regular average-value problems for $\mathbf{VASS}(\mathbb{Z})$ and uniform cost functions are decidable in polynomial time.*

3.2 Natural-valued VASS: VASS($\mathbb{N}$)

For natural-valued VASS, we cannot reduce the number of counters to one; we need to track all counters to make sure that all of them have non-negative values. Moreover, we show that the average-value problem for (single counter) $\mathbf{VASS}(\mathbb{N}, 1)$ is in PSPACE while the average-value problem for $\mathbf{VASS}(\mathbb{N})$ is nonelementary.

First, the average-value problem for (single counter) $\mathbf{VASS}(\mathbb{N}, 1)$ in the uniform case is equivalent to single-player average energy games with non-negativity constraint on the energy values. The latter problem is in PSPACE and it is NP-hard [3].

In the multi-counter case, we show that the average-value problem is mutually reducible to the configuration reachability problem for VASS, which has recently been shown nonelementary hard [10]. We additionally assume that the cost function depends on all its arguments, i.e., all coefficients are non-negative. This assumption allows us show that if there is a computation of the average value below some λ, then there is one, which is lasso-shaped and the cycle in the lasso has exponential size in the size of the VASS and λ. Therefore, we can non-deterministically pick such a cycle and check whether it is reachable from the initial configuration.

▶ **Theorem 4.** *The average-value and the regular average-value problems for $\mathbf{VASS}(\mathbb{N})$ and uniform cost functions with non-zero coefficients are decidable and mutually reducible to the configuration reachability problem for $\mathbf{VASS}(\mathbb{N})$.*

We have used in the proof the fact that f depends on all counters. We conjecture that this assumption can be lifted:

▶ **Open question 5.** *Is the average-value problem for $\mathbf{VASS}(\mathbb{N})$ and uniform cost functions decidable?*

4 General cost functions and VASS($\mathbb{Z}$)

First, we consider (an extension of) the regular finite-value problem for $\mathbf{VASS}(\mathbb{Z})$ (Section 4.1). We show that one can decide in non-deterministic polynomial time whether a given VASS has a regular computation (a) of the value $-\infty$, or (b) of some finite value. To achieve this, we introduce path summarizations, which allow us to state conditions that entail (a) and respectively (b). We show that these conditions can be checked in NP.

We apply the results from Section 4.1 to solve the regular average-value problem (Section 4.2). We consider only VASS that have some computation of a finite value, and no computation of the value $-\infty$, i.e., the answer to (a) is NO and the answer to (b) is YES. In other cases we can easily answer to the regular average-value problem. If the answer to (a) is YES, then for any threshold we answer YES. If the answers to (a) and (b) are NO, then for any threshold we answer NO. Finally, we show NP-hardness of the regular finite-value and the regular average-value problems (Section 4.3).

The main result of this section is the following theorem:

▶ **Theorem 6.** *The regular finite-value and the regular average-value problems for* $\boldsymbol{VASS}(\mathbb{Z})$ *with (general) cost functions are* NP*-complete.*

We fix a VASS$(\mathbb{Z}, k)$ $\mathcal{A} = \langle Q, Q_0, \delta \rangle$, with the set of states Q and the set of transitions δ, and a cost function f, which we refer to throughout this section. We exclude the complexity statements, where the asymptotic behavior applies to all $\mathcal{A}$ and f.

4.1 The finite-value problem

For a path ρ, we define characteristics $\textsc{Gain}(\rho), \textsc{Vals}(\rho)$, which summarize the impact of ρ on the values of counters ($\textsc{Gain}$) and the value of the (partial) average of costs ($\textsc{Vals}$).

Let ρ be of the form $(q_1, q_2, \vec{y}_1) \ldots (q_m, q_{m+1}, \vec{y}_m)$. We define $\textsc{Gain}(\rho)$ as the sum of updates along ρ, i.e., $\textsc{Gain}(\rho) = \sum_{i=1}^{m} \vec{y}_i$. The vector $\textsc{Gain}(\rho)$ is the update of counters upon the whole path ρ. Observe that $\textsc{Gain}(\rho_1\rho_2) = \textsc{Gain}(\rho_1) + \textsc{Gain}(\rho_2)$.

Let $l \colon Q \to \mathbb{N}^k$ be the function representing f, i.e., for every $\vec{z} \in \mathbb{Z}^k$ we have $f(q, \vec{z}) = l(q) \cdot \vec{z}$. We define $\textsc{Vals}(\rho)$ as the sum of vectors $l(q_i)$ along ρ, i.e., $\textsc{Vals}(\rho) = \sum_{i=1}^{m} l(q_i)$. Note that we exclude the last state q_{m+1}, and hence we have $\textsc{Vals}(\rho_1\rho_2) = \textsc{Vals}(\rho_1) + \textsc{Vals}(\rho_2)$. The vector $\textsc{Vals}(\rho)$ describes the coefficients with which each counter contributes to the average along the path ρ.

Consider a regular computation π and a path $\rho_1(\rho_2)^\omega$ that corresponds to it. Let π' be the (regular) computation obtained from π by contracting $|\rho_2|$ to a single transition $(q_f, q_f, \textsc{Gain}(\rho_2))$, which is a loop over a fresh state q_f. The counters over this transition are updated by $\textsc{Gain}(\rho_2)$ and the cost function in q_f is defined as $f(q_f, \vec{z}) = \frac{1}{|\rho_2|}\textsc{Vals}(\rho_2) \cdot \vec{z}$.

The values of π and π' may be different, but they differ only by some finite value. Indeed, the difference over a single iteration of ρ_2 updates and computation of the partial averages are interleaved along ρ_2, while in $(q_f, q_f, \textsc{Gain}(\rho_2))$ we first compute the partial average and then update the counters. Therefore, that difference is a finite value N that does not depend on the initial values of counters. It follows that the difference between the average values of (the computations corresponding to) ρ_2 and $(q_f, q_f, \textsc{Gain}(\rho_2))$ is bounded by N.

Finally, observe that the value of π' can be easily estimated. Observe that the partial average of the first $n+1$ values of f in π' equals

$$\frac{1}{n+1}\Big(\vec{0} \cdot \textsc{Vals}(\rho_2) + \textsc{Gain}(\rho_2) \cdot \textsc{Vals}(\rho_2) + \ldots + n\textsc{Gain}(\rho_2) \cdot \textsc{Vals}(\rho_2)\Big) = \frac{n}{2}\textsc{Gain}(\rho_2) \cdot \textsc{Vals}(\rho_2).$$

In consequence, we have the following:

▶ **Lemma 7.** *Let π be a regular computation corresponding to a path $\rho_1(\rho_2)^\omega$. Then, one of the following holds:*

1. $\textsc{Gain}(\rho_2) \cdot \textsc{Vals}(\rho_2) < 0$ *and* $\textsc{LimAvg}(f(\pi)) = -\infty$*, or*
2. $\textsc{Gain}(\rho_2) \cdot \textsc{Vals}(\rho_2) = 0$ *and* $\textsc{LimAvg}(f(\pi))$ *is finite, or*
3. $\textsc{Gain}(\rho_2) \cdot \textsc{Vals}(\rho_2) > 0$ *and* $\textsc{LimAvg}(f(\pi)) = \infty$*.*

▶ **Example 8.** Consider the VASS $\mathcal{A}_e$ and the cost function f from Example 2. We have shown that the computation defined by the path $(e_1e_2e_3e_4)^\omega$ has finite average value. This can be algorithmically computed using Lemma 7. We compute

$$\textsc{Gain}(e_1e_2e_3e_4) = \binom{1}{0} + \binom{0}{-1} + \binom{0}{3} + \binom{-2}{0} = \binom{-1}{2}$$
$$\textsc{Vals}(e_1e_2e_3e_4) = \binom{1}{1} + \binom{4}{0} + \binom{1}{1} + \binom{0}{1} = \binom{6}{3}$$

Therefore, $\textsc{Gain}(e_1e_2e_3e_4) \cdot \textsc{Vals}(e_1e_2e_3e_4) = 0$.

Reduction to integer quadratic programming. Lemma 7 reduces the regular finite-value problem to finding a cycle with $\textsc{Gain}(\rho) \cdot \textsc{Vals}(\rho) < 0$ (resp., $\textsc{Gain}(\rho) \cdot \textsc{Vals}(\rho) = 0$). We show that existence of such a cycle can be stated as a polynomial-size instance of *integer quadratic programming*, which can be decided in NP [11]. An instance of integer quadratic programming consists of a symmetric matrix $\mathbf{A} \in \mathbb{Q}^{n\times n}, \vec{a} \in \mathbb{Q}^n, d \in \mathbb{Q}, \mathbf{B} \in \mathbb{Q}^{n\times m}, \vec{c} \in \mathbb{Q}^m$ and it asks whether there exists a vector $\vec{x} \in \mathbb{Z}^n$ satisfying the following system:

$$\begin{aligned} \vec{x}^T\mathbf{A}\vec{x} + \vec{a}\vec{x} + d &\leq 0 \\ \mathbf{B}\vec{x} &\leq \vec{c} \end{aligned}$$

Let ρ be a cycle. Observe that both $\textsc{Gain}(\rho), \textsc{Vals}(\rho)$ depend only on the multiplicity of transitions occurring in ρ; the order of transitions is irrelevant. Consider a vector $\vec{x} = (x[1], \ldots, x[m])$ of multiplicities of transitions $e_1, \ldots, e_m$ in ρ. Then,

$$\textsc{Gain}(\rho) \cdot \textsc{Vals}(\rho) = \sum_{1\leq i,j\leq m} x[i]x[j]\textsc{Gain}(e_i) \cdot \textsc{Vals}(e_j)$$

Therefore, we have

$$2 \cdot \textsc{Gain}(\rho) \cdot \textsc{Vals}(\rho) = \vec{x}^T\mathbf{A}\vec{x} \tag{1}$$

for a symmetric matrix $\mathbf{A} \in \mathbb{Z}^{m\times m}$ defined for all i, j as

$$\mathbf{A}[i,j] = \textsc{Gain}(e_i) \cdot \textsc{Vals}(e_j) + \textsc{Gain}(e_j) \cdot \textsc{Vals}(e_i). \tag{2}$$

The left hand side of (1) is multiplied by 2 to avoid division by 2. It follows that if $\textsc{Gain}(\rho) \cdot \textsc{Vals}(\rho) < 0$ (resp., $\textsc{Gain}(\rho) \cdot \textsc{Vals}(\rho) = 0$), then the inequality $\vec{x}^T\mathbf{A}\vec{x} - 1 \leq 0$ (resp., $\vec{x}^T\mathbf{A}\vec{x} \leq 0$) has a solution.

Note that the above inequality can have a solution, which does not correspond to a cycle. We can encode with a system of linear inequalities that $\vec{x}$ corresponds to a cycle. Consider $S \subseteq Q$, which corresponds to all states visited by ρ. It suffices to ensure that (a) for all $s \in S$, the number of incoming transitions to s, which is the sum of multiplicities of transitions leading to s, equals the number of transitions outgoing from s, (b) for every $s \in S$, the number of outgoing transitions is greater or equal to 1, and (c) for $s \in Q \setminus S$, the number of incoming and outgoing transitions is 0. Finally, we state that all multiplicities are non-negative. We can encode such equations and inequalities as $\mathbf{B}_S\vec{x} \leq \vec{c}_S$. Observe that every vector of multiplicities $\vec{x} \in \mathbb{Z}$ satisfies $\vec{x}^T\mathbf{A}\vec{x} - 1 \leq 0$ (resp., $\vec{x}^T\mathbf{A}\vec{x} \leq 0$) and $\mathbf{B}_S\vec{x} \leq \vec{c}_S$ defines a cycle ρ with $\textsc{Gain}(\rho), \textsc{Vals}(\rho) < 0$ (resp., $\textsc{Gain}(\rho), \textsc{Vals}(\rho) = 0$). The matrices $\mathbf{A}, \mathbf{B}_S$ and the vector $\vec{c}_S$ are polynomial in $|\mathcal{A}|$. The set S can be picked non-deterministically. In consequence, we have the following:

▶ **Lemma 9.** *The problem: given a* VASS$(\mathbb{Z}, k)$ $\mathcal{A}$ *and a cost function* f*, decide whether* $\mathcal{A}$ *has a regular computation of the value* $-\infty$ *(resp., less than* $+\infty$*) is in* NP.

4.2 The regular average-value problem

In this section, we study the regular average-value problem for $\mathbf{VASS}(\mathbb{Z})$. We assume that $\mathcal{A}$ has a regular computation of finite value and does not have a regular computation of the value $-\infty$. Finally, we consider the case of the threshold $\lambda = 0$. The case of an arbitrary threshold $\lambda \in \mathbb{Q}$ can be easily reduced to the case $\lambda = 0$ (see [7, 6] for intuitions and techniques in mean-payoff games a.k.a. limit-average games).

Consider a regular computation π and let $\rho_1(\rho_2)^\omega$ be a path corresponding to π. We derive the necessary and sufficient conditions on ρ_1, ρ_2 to have $\textsc{LimAvg}(f(\pi)) \leq 0$.

Assume that $\textsc{LimAvg}(f(\pi)) \leq 0$. Due to Lemma 7, we have $\textsc{Gain}(\rho_2) \cdot \textsc{Vals}(\rho_2) = 0$, which implies that every iteration of ρ_2 has the same average. Therefore, $\textsc{LimAvg}(f(\pi))$ is the average value of ρ_2 (starting with counter values defined by ρ_1). The latter value is at most 0 if and only if the sum of values along ρ_2 is at most 0. We define this sum below.

The sum of values $\mathsf{Sum}_{\vec{g}}(\rho)$. Let l be the labeling defining the cost function f. Consider a path ρ of length m such that $\rho = (q_1, q_2, \vec{y}_1) \dots (q_m, q_{m+1}, \vec{y}_m)$. Let π be the computation corresponding to ρ with the initial counter values being $\vec{g}$. Then, the value of counters at the position i in π is $\vec{g} + \sum_{j=1}^{i-1} \vec{y}_j$. Therefore, the sum of values over ρ starting with counter values $\vec{g} \in \mathbb{Z}^k$, denoted by $\mathsf{Sum}_{\vec{g}}(\rho)$, is given by the following formula:

$$\mathsf{Sum}_{\vec{g}}(\rho) = \sum_{i=1}^{m} \left(\vec{g} + \sum_{j=1}^{i-1} \vec{y}_j \right) \cdot l(q_i)$$

We have the following:

▶ **Lemma 10.** *There exists a regular computation π with $\textsc{LimAvg}(f(\pi)) \leq 0$ if and only if there exist paths ρ_1, ρ_2 such that*
(C1) *$\rho_1(\rho_2)^\omega$ is an infinite path and ρ_2 is a cycle,*
(C2) *$\textsc{Gain}(\rho_2) \cdot \textsc{Vals}(\rho_2) = 0$, and*
(C3) *$\mathsf{Sum}_{\textsc{Gain}(\rho_1)}(\rho_2) \leq 0$.*

Due to results from the previous section, we can check in NP the existence of ρ_1, ρ_2 satisfying (C1) and (C2). We call a cycle ρ_2 *balanced* if $\textsc{Gain}(\rho_2) \cdot \textsc{Vals}(\rho_2) = 0$. In the remaining part we focus on condition (C3), while keeping in mind (C1) and (C2).

The plan of the proof. The proof has three key ingredients which are as follows: *factorizations*, *quadratic factor elimination*, and *the linear case.*

- *Factorizations.* We show that we can consider only paths ρ_2 of the form

  $$\alpha_0 \beta_1^{n[1]} \alpha_1 \beta_2^{n[2]} \dots \beta_p^{n[p]} \alpha_p,$$

  where $|\alpha_i|, |\beta_i| \leq |\mathcal{A}|$ and each β_i is a distinct simple cycle (Lemma 12). It follows that p is exponentially bounded in $|\mathcal{A}|$. Next, we show that for ρ_2 in such a form we have

  $$\mathsf{Sum}_{\textsc{Gain}(\rho_1)}(\rho_2) = \vec{n}^T \mathbf{B} \vec{n} + \vec{c} \cdot \vec{n} + e$$

  where $\mathbf{B} \in \mathbb{Z}^{p \times p}, \vec{c} \in \mathbb{Z}^p$ (Lemma 13). Therefore, $\mathsf{Sum}_{\textsc{Gain}(\rho_1)}(\rho_2) \leq 0$ can be presented as an instance of integer quadratic programming, where the variables correspond to multiplicities of simple cycles.
  However, there are two problems to overcome. First, ρ_2 has to be balanced, i.e., it has to satisfy $\textsc{Gain}(\rho_2) \cdot \textsc{Vals}(\rho_2) = 0$, which introduces another quadratic equation. Solving a system of two quadratic equations over integers is considerably more difficult (see [11] for references). Second, p is (bounded by) the number of distinct simple cycles and hence it can be exponential in $|\mathcal{A}|$. Therefore, $\mathbf{B}$ and $\vec{c}$ may have exponential size (of the binary representation) in $|\mathcal{A}|$.
- *Quadratic factor elimination.* To solve these problems, we fix a sequence $\textsc{Tpl} = (\alpha_0, \beta_1, \alpha_1, \dots, \beta_p, \alpha_p)$ and consider $\mathbf{B}_{\textsc{Tpl}}, \vec{c}_{\textsc{Tpl}}, e_{\textsc{Tpl}}$ for $\textsc{Tpl}$. We show that one of the following holds (Lemma 14 and Lemma 15):

  - $\vec{n}^T \mathbf{B}_{\text{TPL}} \vec{n} + \vec{c}_{\text{TPL}} \cdot \vec{n} + e_{\text{TPL}} \leq 0$ has a simple solution, or
  - there exist $\vec{d}_{\text{TPL}} \in \mathbb{Z}^p, h_{\text{TPL}} \in \mathbb{Z}$ such that for all vectors $\vec{n}$, if the cycle $\rho_2 = \alpha_0 \beta_1^{n[1]} \ldots \beta_p^{n[p]} \alpha_p$ is balanced (C2), then $\vec{n}^T \mathbf{B}_{\text{TPL}} \vec{n} = \vec{d}_{\text{TPL}} \cdot \vec{n} + h_{\text{TPL}}$.

  We can decide in non-deterministic polynomial time whether the first condition holds (Lemma 20).

- *The linear case.* Assuming that the second condition holds, we reduce the problem of solving the quadratic inequality $\vec{n}^T \mathbf{B}_{\text{TPL}} \vec{n} + \vec{c}_{\text{TPL}} \cdot \vec{n} + e_{\text{TPL}} \leq 0$ to solving the linear inequality $(\vec{c}_{\text{TPL}} + \vec{d}_{\text{TPL}}) \cdot \vec{n} + (e_{\text{TPL}} + h_{\text{TPL}}) \leq 0$. Moreover, we can compute $\vec{d}_{\text{TPL}}, h_{\text{TPL}}$ from the sequence TPL. At this point, we can solve the problem in non-deterministic exponential time. Next, we argue that we do not have to compute the whole system. We show that if $(\vec{c}_{\text{TPL}} + \vec{d}_{\text{TPL}}) \cdot \vec{n} + e_{\text{TPL}} + h_{\text{TPL}} \leq 0$ has a solution, then it has a solution for a vector $\vec{n}_0$ with $m = O(|\mathcal{A}|)$ non-zero components. Therefore, we can remove cycles corresponding to 0 coefficients of $\vec{n}_0$. Still, $\sum_{i=0}^{p} |\alpha_i|$ can be exponential in $|\mathcal{A}|$, but this operation shortens the size of the template, i.e., the value $\sum_{i=0}^{p} |\alpha_i| + \sum_{i=1}^{p} |\beta_i|$, and hence by iterating it we get a polynomial size template, which yields a polynomial-size system of inequalities. These inequalities can be solved in NP.

We now present the details of each ingredient.

4.2.1 Factorizations

The regular finite-value problem has been solved via reduction to solving (quadratic and linear) inequalities. In this section, we show a reduction of the regular average-value problem to linear and quadratic inequalities as well. First, we establish that we can consider only cycles ρ, which have a compact representation using *templates* parametrized by multiplicities of cycles. The value $\mathsf{Sum}_{\vec{g}}(\rho)$ for a cycle represented by a template is given by a quadratic function in the multiplicities of cycles.

Templates and multiplicities. A *template* TPL is a sequence of paths $(\alpha_0, \beta_1, \alpha_1, \ldots, \beta_p, \alpha_p)$ such that all $\beta_1, \ldots, \beta_p$ are cycles and $\alpha_0 \beta_1 \alpha_1 \ldots \beta_p \alpha_p$ is a cycle. A template is *minimal* if for all $i \in \{0, \ldots, p\}$ we have $|\alpha_i| < |Q|$ and all β_i are pairwise distinct simple cycles. For every vector $\vec{n} \in \mathbb{N}^p$, called *multiplicities*, we define $\text{TPL}(\vec{n})$ as a cycle $\alpha_0 \beta_1^{n[1]} \alpha_1 \beta_2^{n[2]} \ldots \beta_p^{n[p]} \alpha_p$. A cycle ρ has a (minimal) *factorization* if there exists a (minimal) template and multiplicities $\vec{n}$ such that $\rho = \text{TPL}(\vec{n})$.

Observe that every cycle ρ has a factorization such that for all i we have $|\alpha_i| < |Q|$ and each β_i is a simple cycle. However, the sequence β_i's can have repetitions. The following lemma states that if a cycle β occurs twice in ρ, then we can group them together.

► **Lemma 11.** *Consider $\vec{g} \in \mathbb{Z}^k$ and a cycle $\alpha_0 \beta \alpha_1 \beta \alpha_2$. Then, one of the following holds:* $\mathsf{Sum}_{\vec{g}}(\alpha_0 \beta^2 \alpha_1 \alpha_2) \leq \mathsf{Sum}_{\vec{g}}(\alpha_0 \beta \alpha_1 \beta \alpha_2)$ *or* $\mathsf{Sum}_{\vec{g}}(\alpha_0 \alpha_1 \beta^2 \alpha_2) \leq \mathsf{Sum}_{\vec{g}}(\alpha_0 \beta \alpha_1 \beta \alpha_2)$.

Careful repeated application of Lemma 11 implies that we can look for a cycle ρ satisfying (C1), (C2) and (C3) among cycles that have a minimal factorization:

► **Lemma 12.** *For every cycle ρ and $\vec{g} \in \mathbb{Z}^k$, there exists a cycle ρ' that has a minimal factorization such that* $\text{GAIN}(\rho) = \text{GAIN}(\rho')$, $\text{VALS}(\rho) = \text{VALS}(\rho')$ *and* $\mathsf{Sum}_{\vec{g}}(\rho) \geq \mathsf{Sum}_{\vec{g}}(\rho')$.

Consider a template $\text{TPL} = (\alpha_0, \beta_1, \alpha_1, \ldots, \beta_p, \alpha_p)$. We present $\mathsf{Sum}_{\vec{g}}(\text{TPL}(\vec{n}))$ as a function from multiplicities of simple cycles $\vec{n} \in \mathbb{N}$ into $\mathbb{Z}$. First, observe that

$$\mathsf{Sum}_{\vec{g}}(\text{TPL}(\vec{n})) = \mathsf{Sum}_{\vec{0}}(\text{TPL}(\vec{n})) + \vec{g} \cdot \text{VALS}(\text{TPL}(\vec{n})).$$

The expression $\textsc{Vals}(\textsc{Tpl}(\vec{n}))$ is a linear expression in $\vec{n}$ with natural coefficients, and the expression $\mathsf{Sum}_{\vec{0}}(\textsc{Tpl}(\vec{n}))$ is a quadratic function in each of its arguments $\vec{n}$.

▶ **Lemma 13.** *Given a template* $\textsc{Tpl}$ *we can compute in polynomial time in* $|\textsc{Tpl}|+|\mathcal{A}|+|f|$*, a symmetric matrix* $\mathbf{B}_{\textsc{Tpl}} \in \mathbb{Z}^{p\times p}, \vec{c}_{\textsc{Tpl}} \in \mathbb{Z}^p$ *and* $e_{\textsc{Tpl}} \in \mathbb{Z}$ *such that the following holds:*

$$2 \cdot \mathsf{Sum}_{\vec{0}}(\textsc{Tpl}(\vec{n})) = \vec{n}^T \mathbf{B}_{\textsc{Tpl}} \vec{n} + \vec{c}_{\textsc{Tpl}} \vec{n} + e_{\textsc{Tpl}} \tag{3}$$

Moreover, for all $i, j \in \{1, \ldots, p\}$ *we have*

$$\mathbf{B}_{\textsc{Tpl}}[i,j] = \textsc{Gain}(\beta_{min(i,j)}) \cdot \textsc{Vals}(\beta_{max(i,j)}). \tag{4}$$

Observe that $\mathbf{B}_{\textsc{Tpl}}$ is similar to the matrix $\mathbf{A}$ from (1). We exploit this similarity in the following section to eliminate the term $\vec{n}^T \mathbf{B}_{\textsc{Tpl}} \vec{n}$ if possible.

4.2.2 Elimination of the quadratic factor

We show how to simplify the expression (3) of Lemma 13 for $\mathsf{Sum}_{\vec{0}}(\textsc{Tpl}(\vec{n}))$. We show that either the inequality $\mathsf{Sum}_{\vec{0}}(\textsc{Tpl}(\vec{n})) + \vec{g} \cdot \textsc{Vals}(\textsc{Tpl}(\vec{n})) \leq 0$ has a simple solution for every $\vec{g}$, or the quadratic term in (3) of Lemma 13 can be substituted with a linear term.

Negative and linear templates. Consider a template $\textsc{Tpl}$. A template $\textsc{Tpl}$ is *positive* (resp., *negative*) if there exist multiplicities $\vec{n}_1, \vec{n}_2 \in \mathbb{N}^p$ such that

1. $\vec{n}_1^T \mathbf{B}_{\textsc{Tpl}} \vec{n}_1 > 0$ (resp., $\vec{n}_1^T \mathbf{B}_{\textsc{Tpl}} \vec{n}_1 < 0$), and
2. for every $t \in \mathbb{N}^+$, we have $\textsc{Gain}(\textsc{Tpl}(t\vec{n}_1 + \vec{n}_2)) \cdot \textsc{Vals}(\textsc{Tpl}(t\vec{n}_1 + \vec{n}_2)) = 0$.

A template $\textsc{Tpl}$ is *linear* if there exist $\vec{d}_{\textsc{Tpl}} \in \mathbb{Z}^p$ and $h_{\textsc{Tpl}} \in \mathbb{Z}$ such that for all $\vec{n}$, if $\textsc{Gain}(\textsc{Tpl}(\vec{n})) \cdot \textsc{Vals}(\textsc{Tpl}(\vec{n})) = 0$, then $\vec{n}^T \mathbf{B}_{\textsc{Tpl}} \vec{n} = \vec{d}_{\textsc{Tpl}} \cdot \vec{n} + h_{\textsc{Tpl}}$.

We observe that the existence of a negative cycle $\textsc{Tpl}$ implies that $\mathsf{Sum}_{\vec{g}}(\textsc{Tpl}(\vec{n})) \leq 0$ has a solution for every $\vec{g} \in \mathbb{N}^k$, which in turn implies that the answer to the average-value problem is YES. Basically, for ρ^t defined as $\textsc{Tpl}(t\vec{n}_1 + \vec{n}_2)$ and t big enough we can make $\mathsf{Sum}_{\vec{g}}(\rho^t)$ arbitrarily small.

▶ **Lemma 14.** *If there exist a negative template, whose any state is reachable from some initial state, then the answer to the regular average-value problem with threshold* 0 *is YES.*

We show that either there is a template, which is negative or all templates are linear (Lemma 15). Next, we show that we can check in non-deterministic polynomial time whether there exists a negative template (Lemma 20).

▶ **Lemma 15.** *(1) There exists a negative template or all templates are linear. (2) If there exists a positive template* $\textsc{Tpl}$*, then there exists a negative one of the size bounded by* $|\textsc{Tpl}|^2$*. (3) If a template* $\textsc{Tpl}$ *is linear, we can compute* $\vec{d}_{\textsc{Tpl}}, h_{\textsc{Tpl}}$ *in polynomial time in* $|\textsc{Tpl}| + |\mathcal{A}| + |f|$.

Proof ideas. Consider a template $\textsc{Tpl}$ with all connecting paths being empty, i.e., $\textsc{Tpl} = (\epsilon, \beta_1, \ldots, \beta_p, \epsilon)$. Let $\vec{n}$ be a vector of multiplicities such that a cycle $\textsc{Tpl}(\vec{n})$ is balanced, i.e., $\textsc{Gain}(\textsc{Tpl}(\vec{n})) \cdot \textsc{Vals}(\textsc{Tpl}(\vec{n})) = 0$. We consider three cases:

The case $\vec{n}^T \mathbf{B}_{\mathsf{Tpl}} \vec{n} < 0$. Then, $\vec{n}_1 = \vec{n}$ and $\vec{n}_2 = \vec{0}$ witness negativity of $\textsc{Tpl}$.

The case $\vec{n}^T \mathbf{B}_{\mathsf{Tpl}} \vec{n} > 0$. Then, $\textsc{Tpl}$ is positive and we show that the reversed template $\textsc{Tpl}^R = (\epsilon, \beta_p, \ldots, \beta_1, \epsilon)$ is negative. Equality $\textsc{Gain}(\textsc{Tpl}(\vec{n})) \cdot \textsc{Vals}(\textsc{Tpl}(\vec{n})) = 0$ can be stated as a matrix equation $\vec{n}^T \mathbf{A} \vec{n} = 0$, where $\mathbf{A}$ is defined as in (2). We juxtapose (2) and (4), and get that for all i, j we have $\mathbf{A}[i,j] = \mathbf{B}_{\textsc{Tpl}}[i,j] + \mathbf{B}_{\textsc{Tpl}^R}[p-i+1, p-j+1]$. Thus, $0 = \vec{n}^T \mathbf{A} \vec{n} = \vec{n}^T \mathbf{B}_{\textsc{Tpl}} \vec{n} + \vec{n_R}^T \mathbf{B}_{\textsc{Tpl}^R} \vec{n_R}$, where $\vec{n}_R$ is the reversed vector $\vec{n}$. It follows that $\textsc{Tpl}^R$ is negative.

The above two cases fail. Then for all $\vec{n}$, if $\textsc{Gain}(\textsc{Tpl}(\vec{n})) \cdot \textsc{Vals}(\textsc{Tpl}(\vec{n})) = 0$, then $\vec{n}^T \mathbf{B}_{\textsc{Tpl}} \vec{n} = 0$. Therefore, $\textsc{Tpl}$ is linear with $\vec{d}_{\textsc{Tpl}}, h_{\textsc{Tpl}}$ being $\vec{0}$ and 0 respectively.

Essentially the same line of reasoning can be applied if the connecting paths in $\textsc{Tpl}$ are non-empty. However, in that case $\vec{d}_{\textsc{Tpl}}, h_{\textsc{Tpl}}$ may be non-zero. ◀

Now, we discuss how to check whether there exists a negative template. One of the conditions of negativity is that $\textsc{Gain}(\textsc{Tpl}(t\vec{n}_1 + \vec{n}_2)) \cdot \textsc{Vals}(\textsc{Tpl}(t\vec{n}_1 + \vec{n}_2)) = 0$. Recall that $\textsc{Gain}$ and $\textsc{Vals}$ depend on the multiset of transitions, but not on the order of transitions. Therefore, for a given template $\textsc{Tpl}$, we define $\textsc{Trans}_{\textsc{Tpl}}(\vec{n}) \in \mathbb{N}^m$, where $m = |\delta|$, as the vector of multiplicities of transitions in $\textsc{Tpl}(\vec{n})$. We will write $\textsc{Trans}(\vec{n})$ if $\textsc{Tpl}$ is clear from the context.

▶ **Lemma 16.** *Let* $\textsc{Tpl}$ *be a template and let* $\vec{n} \in \mathbb{N}^p$ *be a vector of multiplicities. There exist* $r_1, \ldots, r_\ell \in \mathbb{Q}^+$ *and* $\vec{z}_1, \ldots, \vec{z}_\ell \in \mathbb{N}^p$ *such that (1) supp*$(\vec{z}_i) \leq m$ *(the number of transitions of* $|\mathcal{A}|$*), (2) there exists* $t \in \mathbb{N}^+$ *such that* $\textsc{Trans}(\vec{z}_i) = t \cdot \textsc{Trans}(\vec{n})$*, and (3)* $\vec{n} = \sum_{i=1}^{\ell} r_i \vec{z}_i$*.*

▶ Remark. The condition $\textsc{Trans}(\vec{z}_i) = t \cdot \textsc{Trans}(\vec{n})$ implies that if the cycle $\textsc{Tpl}(\vec{n})$ is balanced, then all the cycles $\textsc{Tpl}(\vec{z}_1), \ldots, \textsc{Tpl}(\vec{z}_\ell)$ are balanced as well.

Proof ideas. First, observe that $\textsc{Trans}$ is a linear function transforming vectors from $\mathbb{N}^p$ into vectors from $\mathbb{N}^m$. The value p can be exponential w.r.t. $m = |\delta|$ and hence we show that each vector from $\mathbb{N}^p$ can be presented as a linear combination over $\mathbb{Q}^+$ of vectors with polynomially-bounded supports. ◀

Next, we show that if there exists a negative template, then there exists one of polynomial size. If $\vec{n}_1, \vec{n}_2 \in \mathbb{N}^p$ are the vectors witnessing positivity (resp., negativity) of a template $\textsc{Tpl}$, then by Lemma 16, there exist witnesses $\vec{n}_1^0, \vec{n}_2^0$ with polynomial-size support. We remove from $\textsc{Tpl}$ cycles corresponding to coefficient 0 in both $\vec{n}_1^0$ and $\vec{n}_2^0$ and obtain only polynomially many cycles. In consequence, we have:

▶ **Lemma 17.** *If there exists a negative template, then there exists one of polynomial size in* $|\textsc{Tpl}| + |\mathcal{A}| + |f|$*.*

Still, to check whether there exists a negative template we have to solve a system consisting of a quadratic inequality $\vec{n}_1^T \mathbf{B}_{\textsc{Tpl}} \vec{n}_1 < 0$ and a quadratic equation, which corresponds to $\textsc{Gain}(\textsc{Tpl}(t\vec{n}_1 + \vec{n}_2)) \cdot \textsc{Vals}(\textsc{Tpl}(t\vec{n}_1 + \vec{n}_2)) = 0$. We show that this quadratic equation can be transformed into a system of linear inequalities. We show that using standard elimination of quadratic terms for successive variables $n[1], n[2], \ldots, n[p+1]$. The key observation is that $\textsc{Gain}(\textsc{Tpl}(\vec{n})) \cdot \textsc{Vals}(\textsc{Tpl}(\vec{n})) = 0$ implies that for every variable $n[i]$ either $n[i] = 0$ or $n[i]$ is a double root ($\Delta = 0$). Thanks to this property, we obtain a polynomial-size system of linear inequalities.

▶ **Lemma 18.** *Let* $\textsc{Tpl}$ *be a template. There exist systems of linear equations and inequalities* $S_1, \ldots, S_l$ *such that (1) each* S_i *has polynomial size in* $|\textsc{Tpl}| + |\mathcal{A}| + |f|$*, (2) for all* $\vec{n} \in \mathbb{N}^p$ *we have* $\textsc{Gain}(\textsc{Tpl}(\vec{n})) \cdot \textsc{Vals}(\textsc{Tpl}(\vec{n})) = 0$ *iff for some* i *the vector* $\vec{n}$ *satisfies* S_i*.*

Finally, Lemma 17 and Lemma 18 imply that existence of a negative template can be solved in NP via reduction to integer quadratic programming.

▶ **Lemma 19.** *We can verify in* NP *whether a given* $\text{VASS}(\mathbb{Z}, k)$ $\mathcal{A}$ *has a negative template.*

Proof sketch. We non-deterministically pick a template $\textsc{Tpl}$ of polynomial size (Lemma 17). Then, we non-deterministically pick a system S_i for template (Lemma 18). We make two copies of S_i: S_i^1 and S_i^2; in one we substitute $\vec{n}$ with $\vec{n}_2$ and in the other with $\vec{n}_1 + \vec{n}_2$. It follows that for all $t \in \mathbb{R}$ the vector $t\vec{n}_1 + \vec{n}_2$ satisfies S_i (substituted for $\vec{n}$). Finally, we solve an instance of integer quadratic programming consisting of $\vec{n}_1 \mathbf{B}_{\textsc{Tpl}} \vec{n}_1 - 1 \leq 0$ and linear equations and inequalities S_i^1 and S_i^2, which is an instance of integer quadratic programming and hence can be solved in NP [11]. ◀

4.2.3 The linear case

We consider the final case, where all templates are linear. The decision procedure described in the following lemma answers YES (in at least one of non-deterministic computations) whenever the answer to the regular average-value problem with threshold 0 is YES and all templates are linear. Thus, it is complete.

Our main algorithm assumes that all templates are linear if it fails to find a negative template. The failure can be due to a wrong non-deterministic pick. Having that in mind, we make sure that the decision procedure from the following lemma is sound regardless of the linearity of templates, i.e., if it answers YES, then the answer to the regular average-value problem with threshold 0 is YES.

▶ **Lemma 20.** *Assume that all templates are linear. Then, we can solve the regular average-value problem with threshold* 0 *in non-deterministic polynomial time. Moreover, the procedure is sound irrespectively of the linearity assumption.*

Proof sketch. First, we show that using Lemma 16, if there is a cycle ρ with (a) $\textsc{Gain}(\rho) \cdot \textsc{Vals}(\rho) = 0$ and (b) $\mathsf{Sum}_{\vec{g}}(\rho) \leq 0$, then there is a cycle ρ' defined by a template of polynomial size satisfying both conditions (a) and (b). We take a path ρ, write it as $\textsc{Tpl}(\vec{n})$ and apply Lemma 16 to $\vec{n}$. We get a vector with polynomially many non-zero coefficients $\vec{n}_0$ and define a reduced template $\textsc{Tpl}'$ by removal of cycles that correspond to 0 coefficients.

Second, consider a template $\textsc{Tpl}$ of polynomial size. We nondeterministically pick a subset of states Q and write a system of equations $S^Q_{\textsc{Gain}}$ over variables $\vec{x}, \vec{y}$ such that $\vec{x}, \vec{y}$ is a solution of $S^Q_{\textsc{Gain}}$ if and only if there exists a path ρ_1 satisfying (a) $\vec{x}$ are multiplicities of transitions along ρ_1 (b) ρ_1 is from some initial state of $\mathcal{A}$ to the first state of $\textsc{Tpl}$, (c) ρ_1 visits all states from Q, (d) $\vec{y} = \textsc{Gain}(\rho_1)$.

Finally, if all templates are linear, then there exist ρ_1, ρ_2 defining a regular computation of the average value at most 0 if and only if there is a subset of states Q and a template $\textsc{Tpl}$ of polynomial size such that the system of inequalities consisting of $S^Q_{\textsc{Gain}}$ and the inequality $H_{x,y} : (\vec{c}_{\textsc{Tpl},\vec{0}} + \vec{d}_{\textsc{Tpl}}) \cdot \vec{n} + e_{\textsc{Tpl},\vec{0}} + h_{\textsc{Tpl}} + 2 \cdot \vec{y} \cdot \textsc{Vals}(\textsc{Tpl}(\vec{n})) \leq 0$ has a solution over natural numbers. Note that all the components except for $\vec{y} \cdot \textsc{Vals}(\textsc{Tpl}(\vec{n}))$ are linear. Since $\vec{y}$ and $\vec{n}$ are variables, the component $\vec{y} \cdot \textsc{Vals}(\textsc{Tpl}(\vec{n}))$ is quadratic. Still, $S^Q_{\textsc{Gain}}$ with $H_{x,y}$ is an instance of integer quadratic programming, which can be solved in NP [11].

Having a solution of the system $S^Q_{\textsc{Gain}}$, we can compute in polynomial time $\textsc{Gain}(\rho_1)$ and $\mathsf{Sum}_{\textsc{Gain}(\rho_1)}(\textsc{Tpl}(\vec{n}))$ and then verify $\mathsf{Sum}_{\textsc{Gain}(\rho_1)}(\textsc{Tpl}(\vec{n})) \leq 0$. Therefore, the correctness of the algorithm does not depend on the linearity assumption. ◀

4.2.4 Summary

We present a short summary of the non-deterministic procedure deciding whether a given $\mathrm{VASS}(\mathbb{Z}, k)$ $\mathcal{A}$ has a regular computation of the value at most 0. We assume that all states in $\mathcal{A}$ are reachable from initial states.

- *Step 1.* Check whether there is a cycle ρ with $\textsc{Gain}(\rho) \cdot \textsc{Vals}(\rho) < 0$. It can be done in non-deterministic polynomial time (Lemma 9). If the answer is YES, then the answer to the average-value problem is YES (Lemma 7). Otherwise, proceed to Step 2.
- *Step 2.* Check whether there exists a negative template in $\mathcal{A}$. It can be done in non-deterministic polynomial time (Lemma 19). If the answer is YES, then the answer to the regular average-value problem is YES (Lemma 14). Otherwise, proceed to Step 3.
- *Step 3.* Assuming that the previous steps failed, all templates are linear. Solve the regular average-value problem in non-deterministic polynomial time (Lemma 20). Note that Step 2, could have failed due to unfortunate non-deterministic pick. However, the procedure from Lemma 20 is sound regardless of the linearity assumption.

In consequence, we have the main result of this section:

▶ **Lemma 21.** *The regular average-value problem for **VASS**$(\mathbb{Z})$ with (general) cost functions is in* NP.

4.3 Hardness

We show that the regular finite-value and the regular average-value problems are NP-hard. The proof is via reduction from the 3-SAT problem. Given a 3-CNF formula φ over n variables, we construct a VASS of dimension $2n$, where dimensions correspond to literals in φ. Each simple cycle ρ in the VASS consists of two parts: The first part corresponds to picking a substitution σ, which is stored in the vector $\textsc{Gain}(\rho)$. The second part ensures that $\textsc{Gain}(\rho) \cdot \textsc{Vals}(\rho) = 0$ if σ satisfies φ and it is strictly positive otherwise. Therefore, if φ is satisfiable, the VASS has a regular run of the average cost 0, and otherwise all its regular runs have infinite average cost. In consequence, we have the following:

▶ **Lemma 22.** *The regular finite-value and the regular average-value problems for **VASS**$(\mathbb{Z})$ with (general) cost functions are* NP*-hard.*

The main results of this section (Lemma 9, Lemma 21 and Lemma 22) summarize to Theorem 6. We leave the case of non-regular runs as an open question.

▶ **Open question 23.** *What is the complexity of the average-value and finite-value problems for **VASS**$(\mathbb{Z})$?*

5 General cost functions and VASS$(\mathbb{N})$

We show that the average-value and the regular average-value problems are undecidable. The proofs are via reduction from the halting problem for Minsky machines [17], which are automata with two natural-valued *registers* r_1, r_2. There are two main differences between Minsky machines and **VASS**$(\mathbb{N})$. First, the former can perform zero- and nonzero-tests on their registers, while the latter can take any transition as long as the counters' values remain non-negative. Second, the halting problem for Minsky machines is qualitative, i.e., the answer is YES or NO. We consider quantitative problems for VASS, where we are interested in the values assigned to computations. We exploit the quantitative features of our problems to simulate zero- and nonzero-tests.

The problems with an exact threshold are undecidable, but we can decide existence of a regular computation of some finite value via reduction to reachability in VASS.

▶ **Theorem 24.** *(1) The average-value and the regular average-value problems for **VASS**$(\mathbb{N})$ with (general) cost functions are undecidable. (2) The regular finite-value problem for **VASS**$(\mathbb{N})$ with (general) cost functions is decidable.*

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Piotr Hofman, Richard Mayr, K. Narayan Kumar, and Patrick Totzke. Infinite-state energy games. In *CSL-LICS 2014*, pages 7:1–7:10, 2014.

2 Roderick Bloem, Swen Jacobs, Ayrat Khalimov, Igor Konnov, Sasha Rubin, Helmut Veith, and Josef Widder. Decidability in Parameterized Verification. *SIGACT News*, 47(2):53–64, 2016.

3 Patricia Bouyer, Nicolas Markey, Mickael Randour, Kim G. Larsen, and Simon Laursen. Average-energy games. *Acta Inf.*, 55(2):91–127, 2018.

4 Tomás Brázdil, Krishnendu Chatterjee, Antonín Kucera, Petr Novotný, Dominik Velan, and Florian Zuleger. Efficient Algorithms for Asymptotic Bounds on Termination Time in VASS. In *LICS 2018*, pages 185–194, 2018.

5 Tomás Brázdil, Stefan Kiefer, Antonín Kucera, and Petr Novotný. Long-Run Average Behaviour of Probabilistic Vector Addition Systems. In *LICS 2015*, pages 44–55, 2015. doi:10.1109/LICS.2015.15.

6 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Nested Weighted Limit-Average Automata of Bounded Width. In *MFCS 2016*, pages 24:1–24:14, 2016.

7 Krishnendu Chatterjee, Thomas A. Henzinger, and Jan Otop. Quantitative Monitor Automata. In *SAS 2016*, pages 23–38, 2016.

8 Krishnendu Chatterjee and Yaron Velner. Hyperplane separation technique for multidimensional mean-payoff games. *J. Comput. Syst. Sci.*, 88:236–259, 2017.

9 Krishnendu Chatterjee and Yaron Velner. The Complexity of Mean-Payoff Pushdown Games. *J. ACM*, 64(5):34:1–34:49, 2017.

10 Wojciech Czerwinski, Slawomir Lasota, Ranko Lazic, Jérôme Leroux, and Filip Mazowiecki. The Reachability Problem for Petri Nets is Not Elementary. In *STOC*, pages 398–406, 2019.

11 Alberto Del Pia, Santanu S Dey, and Marco Molinaro. Mixed-integer quadratic programming is in NP. *Mathematical Programming*, 162(1-2):225–240, 2017.

12 Emanuele D'Osualdo, Jonathan Kochems, and C. H. Luke Ong. Automatic Verification of Erlang-Style Concurrency. In Francesco Logozzo and Manuel Fähndrich, editors, *SAS 2013*, pages 454–476, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. doi:10.1007/978-3-642-38856-9_24.

13 Javier Esparza. Decidability and complexity of Petri net problems—an introduction. *Lectures on Petri nets I: Basic models*, pages 374–428, 1998.

14 Javier Esparza and Mogens Nielsen. Decidability Issues for Petri Nets - a survey. *Bulletin of the European Association for Theoretical Computer Science*, 52:245–262, 1994.

15 Yu Feng, Ruben Martins, Yuepeng Wang, Isil Dillig, and Thomas W. Reps. Component-based Synthesis for Complex APIs. In *POPL 2017*, POPL 2017, pages 599–612, New York, NY, USA, 2017. ACM. doi:10.1145/3009837.3009851.

16 Pierre Ganty and Rupak Majumdar. Algorithmic Verification of Asynchronous Programs. *ACM Trans. Program. Lang. Syst.*, 34(1):6:1–6:48, May 2012. doi:10.1145/2160910.2160915.

17 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.

18 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. Dynamic Cutoff Detection in Parameterized Concurrent Programs. In *CAV 2010*, pages 645–659, 2010. doi:10.1007/978-3-642-14295-6_55.

19 Alexander Kaiser, Daniel Kroening, and Thomas Wahl. Efficient Coverability Analysis by Proof Minimization. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR 2012*, pages 500–515, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. doi:10.1007/978-3-642-32940-1_35.

20 Richard M. Karp and Raymond E. Miller. Parallel Program Schemata. *J. Comput. Syst. Sci.*, 3(2):147–195, 1969.

21 S. Rao Kosaraju. Decidability of Reachability in Vector Addition Systems (Preliminary Version). In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing, May 5-7, 1982, San Francisco, California, USA*, pages 267–281, 1982. doi:10.1145/800070.802201.

22 Jean-Luc Lambert. A Structure to Decide Reachability in Petri Nets. *Theor. Comput. Sci.*, 99(1):79–104, 1992. doi:10.1016/0304-3975(92)90173-D.

23 Jérôme Leroux. Vector addition systems reachability problem (a simpler solution). In *EPiC*, volume 10, pages 214–228. Andrei Voronkov, 2012.

24 Jérôme Leroux. Polynomial Vector Addition Systems With States. In *ICALP 2018*, pages 134:1–134:13, 2018.

25 R. Lipton. The Reachability Problem Requires Exponential Space. Technical report 62, Yale, 1976.

26 Ernst W. Mayr. An Algorithm for the General Petri Net Reachability Problem. In *STOC 1981*, pages 238–246, 1981. doi:10.1145/800076.802477.

27 Jakub Michaliszyn and Jan Otop. Average Stack Cost of Büchi Pushdown Automata. In *FSTTCS 2017*, pages 42:1–42:13, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.42.

28 Charles Rackoff. The covering and boundedness problems for vector addition systems. *Theoretical Computer Science*, 6(2):223–231, 1978. doi:10.1016/0304-3975(78)90036-1.

29 Moritz Sinn, Florian Zuleger, and Helmut Veith. A Simple and Scalable Static Analysis for Bound Analysis and Amortized Complexity Analysis. In *CAV*, pages 745–761, 2014.

Reachability for Bounded Branching VASS

Filip Mazowiecki
LaBRI, Université de Bordeaux, France
filip.mazowiecki@u-bordeaux.fr

Michał Pilipczuk
University of Warsaw, Poland
michal.pilipczuk@mimuw.edu.pl

Abstract

In this paper we consider the reachability problem for bounded branching VASS. Bounded VASS are a variant of the classic VASS model where all values in all configurations are upper bounded by a fixed natural number, encoded in binary in the input. This model gained a lot of attention in 2012 when Haase et al. showed its connections with timed automata. Later in 2013 Fearnley and Jurdziński proved that the reachability problem in this model is PSPACE-complete even in dimension 1. Here, we investigate the complexity of the reachability problem when the model is extended with branching transitions, and we prove that the problem is EXPTIME-complete when the dimension is 2 or larger.

2012 ACM Subject Classification Theory of computation → Models of computation

Keywords and phrases Branching VASS, counter machines, reachability problem, bobrvass

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.28

Related Version An arXiv version of the paper is available at `https://arxiv.org/abs/1904.10226`.

Funding *Filip Mazowiecki*: This study has been carried out with financial support from the French State, managed by the French National Research Agency (ANR) in the frame of the "Investments for the future" Programme IdEx Bordeaux (ANR-10-IDEX-03-02).
Michał Pilipczuk: This work is a part of project TOTAL that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, grant agreement No. 677651.

Acknowledgements The authors would like to thank Marthe Bonamy for feeding them during the work on this project. No beavers were harmed in the making of this paper.

1 Introduction

Vector addition systems with states (VASS or d-VASS when the dimension is d), also known as Petri nets, are a long established model for concurrent systems. The branching generalisation of VASS (BrVASS or d-BrVASS[1]) is among the most popular extensions due to its multiple connections with database theory [4], recursively parallel programs [5], timed pushdown systems [6], and many others. A configuration of a d-VASS (or a d-BrVASS) is a pair consisting of a state and a vector over $\mathbb{N}^d$. In d-VASS, the transitions update the configurations by changing the state and updating the configuration vector using vectors over $\mathbb{Z}^d$. In d-BrVASS, in addition there can be branching transitions, which create two independent copies of the system splitting the configuration vector among them.

[1] The standard abbreviation for this model appearing in the literature is BVASS. Since in this work we work with both bounded and branching models of VASS, we prefer to disambiguate these variants by using prefixes Bo and Br, respectively.

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 28; pp. 28:1–28:13
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Table 1 Complexities in dimension 1. In the last entry only a PSPACE lower bound is known.

	unary	binary	binary with a bound
1-VASS	NL-complete [20]	NP-complete [14]	PSPACE-complete [10]
1-BrVASS	PTIME-complete [13]	PSPACE-complete [11]	in EXPTIME

The central decision problem for both VASS and BrVASS is the *reachability problem*, i.e., whether there exists a computation between given input configurations. For VASS we know that reachability is decidable [18], moreover, recently the upper bound on the complexity was improved to Ackermann [17]. Whether reachability is decidable for BrVASS is one of the most intriguing open problems in formal verification, highlighted for example in the survey of Bojańczyk [3].

To understand better the source of hardness, the reachability problem was studied when the dimension of the vectors is fixed to a constant. Then it matters how the vectors of the models are encoded and one must consider two cases: unary and binary encodings. Particularly, when the dimension is 1 it is known that 1-VASS with vectors encoded in unary is NL-complete [20], while for the binary encoding it becomes NP-complete [14]. For 1-BrVASS the complexities are also known, namely for the unary encoding the reachability is PTIME-complete [13], while for the binary encoding the problem is PSPACE-complete [11]. We present these results in Table 1. For these models the complexities are easy to remember as in both cases the complexity of the branching variant adds alternation to the complexity, as expected. However, branching is not alternation and this intuition is possibly misleading. Already in dimension 2 reachability for 2-VASS is known to be NL-complete [9] and PSPACE-complete [2] for unary and binary encodings, respectively; while for 2-BrVASS it is not even known whether reachability is decidable.

In this paper we are interested in the variants of VASS and BrVASS with vectors encoded in binary, where all values in all configurations are upper bounded by some natural number, given on input. We shall denote these classes BoVASS and BoBrVASS, respectively. The model BoVASS was popularised due to its connections with timed automata [15], where it was shown that reachability for timed automata is interreducible in logarithmic space with reachability for BoVASS. At the time it was known that for timed automata this problem is PSPACE-complete [1], even when there are only 3 clocks available [7], and NL-complete for the case with 1 clock; leaving open the complexity for timed automata with 2 clocks. The problem for two clocks was proved later to be PSPACE-complete [10], where the main technical contribution was proving that reachability for 1-BoVASS is PSPACE-complete. Then the final result for timed automata with two clocks followed from the reductions presented in [15][2]. Regarding the branching extension a more powerful version of the model 1-BoBrVASS was studied in [6], where connections between BrVASS and pushdown timed automata were explored.

The reachability for 1-BoVASS has become one of the standard problems used in reductions proving PSPACE-hardness of various decision problems. Apart from the already mentioned application to prove PSPACE-hardness of reachability for two clock automata, 1-BoVASS was used e.g., to prove: PSPACE-completeness of reachability for two dimensional VASS [2]; and PSPACE-completeness of regular separability of one counter automata [8].

[2] Both papers [15, 10] write *bounded counter automata* instead of BoVASS. We write BoVASS to have a uniform terminology for all models in this paper.

Our contribution. In this paper we study the complexity of the reachability problem for BoBrVASS. This problem can be easily proved to be in EXPTIME, but only PSPACE-hardness is known; e.g., due to the mentioned PSPACE-hardness of 1-BoVASS. Our main contribution is that the reachability problem for 2-BoBrVASS is EXPTIME-complete and we leave the complexity of 1-BoBrVASS as an open problem (see Table 1). We believe that if there exists an EXPTIME lower bound for reachability for 1-BoBrVASS, then this would give a new, interesting starting problem for EXPTIME-hardness reductions. As an example application, we prove that reachability for 1-BoBrVASS is reducible in polynomial time to reachability in 2-BrVASS, which slightly improves the state of art, as currently only a PSPACE lower bound is known for 2-BrVASS [2].

Organisation. After presenting the main definitions and problems in Section 2 and Section 3, we prove the EXPTIME-completeness of reachability for 2-BoBrVASS in Section 4. Then in Section 5 we give some indications that using our techniques it would be hard to lift the EXPTIME-hardness result to 1-BoBrVASS. We conclude in Section 6 providing a polynomial time reduction of reachability for 1-BoBrVASS to reachability for 2-BrVASS.

2 Preliminaries

VASS model. A *Vector Addition System with States* (VASS) in dimension d (in short d-VASS) is $\mathcal{V} = (Q, \Delta)$, where: Q is a finite set of states and $\Delta \subseteq Q \times \mathbb{Z}^d \times Q$ is a finite set of transitions. The size of $\mathcal{V}$ is $|Q| + |\Delta| \cdot r$, where r is the maximal representation size of vectors in Δ (written in binary).

A configuration is a pair $(q, \boldsymbol{n})$, denoted $q(\boldsymbol{n})$, where $\boldsymbol{n} \in \mathbb{N}^d$. Whenever $d = 1$, we identify vectors in dimension 1 with natural numbers and in particular we denote configurations by $q(n)$ for $n \in \mathbb{N}$. We will often refer to values of vector components as counter values and when convenient we will denote them $c_1, \ldots, c_d$. We write that q is the state of the configuration and $\boldsymbol{n}$ is the vector of the configuration. We write $p(\boldsymbol{n}) \xrightarrow{t} q(\boldsymbol{m})$ if $t = (p, \boldsymbol{z}, q) \in \Delta$ and $\boldsymbol{n} + \boldsymbol{z} = \boldsymbol{m}$. More generally, a *run* is a sequence of configurations

$$q_0(\boldsymbol{n}_0) \xrightarrow{t_0} q_1(\boldsymbol{n}_1) \xrightarrow{t_1} \ldots \xrightarrow{t_k} q_{k+1}(\boldsymbol{n}_{k+1}), \tag{1}$$

where $t_i = (q_i, \boldsymbol{z}_i, q_{i+1}) \in \Delta$ and $\boldsymbol{n}_i + \boldsymbol{z}_i = \boldsymbol{n}_{i+1}$ for all $0 \leqslant i \leqslant k$. Notice that configurations have vectors over natural numbers and transitions have vectors over integers. We write $p(\boldsymbol{n}) \xrightarrow{t_0 \ldots t_k} q(\boldsymbol{m})$ or $p(\boldsymbol{n}) \rightarrow^* q(\boldsymbol{m})$ if there exists a run like (1) such that $p(\boldsymbol{n}) = q_0(\boldsymbol{n}_0)$ and $q(\boldsymbol{m}) = q_{k+1}(\boldsymbol{n}_{k+1})$. We write $p(\boldsymbol{n}) \not\rightarrow^* q(\boldsymbol{m})$ if no such run exists.

A bounded d-VASS (d-BoVASS) is a d-VASS equipped with a bound B such that numbers in all configurations cannot exceed B. Formally, a BoVASS $\mathcal{V}$ is a tuple (Q, Δ, B) such that (Q, Δ) is a VASS and $B \in \mathbb{N}$. We will assume that B is given in binary. A run is defined in the same way like for d-VASS but in (1) we additionally assume that $\boldsymbol{n}_i \in \{0, \ldots, B\}^d$ for all $0 \leqslant i \leqslant k$. That is, all components in all vectors are bounded by B. Whenever speaking about bounded models of vector addition systems, we add the length of the bit representation of B to the size of the system.

In the d-VASS model one can simulate lower bound inequality tests as follows. We write that $t = (p, n, i, q) \in Q \times \mathbb{N} \times \{1, \ldots, d\} \times Q$ is an inequality test, and we allow to write $p(\boldsymbol{n}) \xrightarrow{t} q(\boldsymbol{m})$ only if $\boldsymbol{n} = \boldsymbol{m}$ and $c_i \geqslant n$ in both configurations. We use the notation $t = (p, q)_{c_i \geqslant n}$ for simplicity. In a d-VASS $\mathcal{V}$ such a test can be simulated with one extra state r and two transitions: $t_1 = (p, -\boldsymbol{v}_i, r)$ and $t_2 = (r, \boldsymbol{v}_i, q)$, where $\boldsymbol{v}_i[j] = n$ for $i = j$ and $\boldsymbol{v}_i[j] = 0$ otherwise. Then it is easy to see that $p(\boldsymbol{n}) \xrightarrow{t} q(\boldsymbol{m})$ iff $p(\boldsymbol{n}) \xrightarrow{t_1 t_2} q(\boldsymbol{m})$ for all pairs of configurations $p(\boldsymbol{n})$ and $q(\boldsymbol{m})$. If $\mathcal{V}$ is additionally a BoVASS with a bound $B \geqslant n$, then we can also implement analogously defined upper bound inequality tests $(p,q)_{c_i \leqslant n}$. Indeed, it suffices to add one extra state r and two transitions: $t_1 = (p, \boldsymbol{w}_i, r)$ and $t_2 = (r, -\boldsymbol{w}_i, q)$, where $\boldsymbol{w}_i[j] = B - n$ for $i = j$ and $\boldsymbol{w}_i[j] = 0$ otherwise. Finally, equality tests $(p,q)_{c_i = n}$ can be encoded by a lower bound test followed by an upper bound test. Slightly abusing the notation, we will assume that such inequality and equality tests are allowed in the transition set of BoVASS models, as they can be simulated as above at the cost of increasing the size of the system by a constant multiplicative factor.

▶ **Definition 1.** *Fix some function $f\colon \{0,\dots,M\}^d \to \{0,\dots,N\}^d$ for some $M, N \in \mathbb{N}$. We say that a d-*BoVASS *computes f if there exist states $p, q \in Q$ such that for all $\boldsymbol{n} \in \{0,\dots,M\}^d$ and $\boldsymbol{m} \in \mathbb{N}^d$, we have $p(\boldsymbol{n}) \to^* q(\boldsymbol{m})$ if and only if $\boldsymbol{m} = f(\boldsymbol{n})$. The bound B of the* BoVASS *does not have to be equal to M or N.*

Note that in the above definition, we do not specify how the BoVASS computing f behaves starting in a configuration outside of the domain of f. However, it is easy to restrict the system to block on such configurations, e.g. by adding tests $c_i \leqslant M$ in the beginning.

▶ **Example 2.** Consider the copying function $f\colon \{0,\dots,M\}^2 \to \{0,\dots,M\}^2$ defined by $f(n,m) = (n,n)$ for all $0 \leqslant n, m \leqslant M$. We define a 2-BoVASS $\mathcal{V} = (Q, \Delta, B)$ computing f as follows. The bound is $B = M(M+2)$; the set of states is $Q = \{p, r_1, r_2, q\}$; the transitions are depicted in Figure 1. To prove that this is the right function it suffices to observe that:

- $p(n,m) \to^* r_1(k,l)$ iff $k = n$ and $l = 0$;
- $r_1(n,0) \to^* r_2(k,l)$ iff $k = 0$ and $l = (M+2)n$;
- and $r_2(0,(M+2)n) \to^* q(k,l)$ iff $k = l = n$.

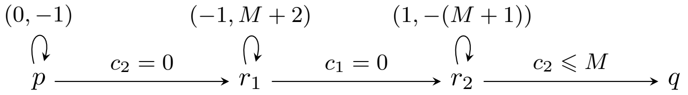


Figure 1 The BoVASS computing $f(n,m) = (n,n)$.

Note that in Example 2, the size of $\mathcal{V}$ is polynomial in the representation of M. A 2-BoVASS of exponential size is trivial to define. In fact, for every function f there exists a BoVASS computing f of size exponential in the representation of the sizes of domain and codomain: one can simply use an equality test for every element of the domain to specify the corresponding value.

Branching VASS model. A *Branching Vector Addition System with States* (BrVASS) in dimension d (in short d-BrVASS) is $\mathcal{B} = (Q, \Delta_1, \Delta_2, q_0)$, where: Q is a finite set of states, $\Delta_1 \subseteq Q \times Z^d \times Q$ is the set of unary transitions, $\Delta_2 \subseteq Q^3$ is the set of binary transitions, and $q_0 \in Q$ is an initial state. The notions regarding configurations are the same as for VASS. A *run* of a BrVASS is a tree labelled with configurations such that internal nodes have either one or two children, and the following conditions hold.

- For every internal node with one child, let $p(\boldsymbol{n})$ be its label and let $q(\boldsymbol{m})$ be the label of its child. Then there exists $(p, \boldsymbol{z}, q) \in \Delta_1$ such that $\boldsymbol{n} + \boldsymbol{z} = \boldsymbol{m}$;
- For every internal node with two children, let $p(\boldsymbol{n})$ be its label and let $q_1(\boldsymbol{m}_1)$, $q_2(\boldsymbol{m}_2)$ be the labels of its children. Then $(p, q_1, q_2) \in \Delta_2$ and $\boldsymbol{n} = \boldsymbol{m}_1 + \boldsymbol{m}_2$.
- The leaves are labelled $q_0(\boldsymbol{0})$, where $\boldsymbol{0}$ is the zero vector.

Notice that a BrVASS with $\Delta_2 = \emptyset$ is essentially a VASS with a distinguished state q_0. A *partial run* is a run that does not have to satisfy the last condition.

A *bounded* d-BrVASS (d-BoBrVASS) is a d-BrVASS equipped with a bound $B \in \mathbb{N}$ that bounds all numbers in all configurations, similarly as for d-BoVASS. As in the case of BoVASS, for every BoBrVASS we abuse the notation allowing for inequality and equality test transitions in Δ_1.

A $(p(\boldsymbol{n}), q(\boldsymbol{m}))$-context (or just context if the configurations are not relevant) of a BrVASS is a partial run such that the root of the tree is labelled $p(\boldsymbol{n})$ and one of the leaves is labelled $q(\boldsymbol{m})$; all other leaves are labelled $q_0(\boldsymbol{0})$. We write $p(\boldsymbol{n}) \rightarrow^* q(\boldsymbol{m})$ if there exists a $(p(\boldsymbol{n}), q(\boldsymbol{m}))$-context and we write $p(\boldsymbol{n}) \not\rightarrow^* q(\boldsymbol{m})$ otherwise. Using this notation, the definition of computing a function $f\colon \{0,\ldots,M\}^d \to \{0,\ldots,N\}^d$ by a d-BoBrVASS generalises from the definition for d-BoVASS in the expected way.

For future reference, we state and prove the following easy claim.

▶ **Lemma 3.** *Given a d-BoBrVASS $\mathcal{B}$ with state set Q and upper bound B together with an integer $B' \geqslant B$, one can compute in polynomial time a d-BoBrVASS $\mathcal{B}'$ with state set $Q' \supseteq Q$ and upper bound B' such that the following holds: for any $p, q \in Q$ and $n, m \in \{0,\ldots,B\}$, we have $p(n) \rightarrow^* q(m)$ in $\mathcal{B}$ if and only if $p(n) \rightarrow^* q(m)$ in $\mathcal{B}'$.*

Proof. It suffices to modify $\mathcal{B}$ as follows: after each transition of $\mathcal{B}$, add an additional sequence of inequality tests checking that all counter values are bounded by B. ◀

The reachability problem. The reachability question for all mentioned models is, given a system within the model (say, a VASS or a BoBrVASS etc.) and two configurations $p(\boldsymbol{n})$, $q(\boldsymbol{m})$, whether $p(\boldsymbol{n}) \rightarrow^* q(\boldsymbol{m})$[3].

In this paper we are interested in the reachability problem for bounded models with vectors encoded in binary. For d-BoVASS (i.e. without branching), the problem is easily contained in PSPACE for any d, because configurations can be described in polynomial space and solving the reachability problem amounts to nondeterministically guessing a run. Fearnley and Jurdziński proved in [10] that, in fact, the problem is PSPACE-complete even for $d = 1$.

For the BoBrVASS model the situation is somewhat similar, however there is a gap in our understanding. On one hand, the problem is trivially in EXPTIME for any d, because the total number of configurations is exponential in the size of the input system. Note here, that even though again every configuration can be described in polynomial space, due to branching we cannot apply the same reasoning as for BoVASS to argue PSPACE membership. For lower bounds, the result of [10] shows PSPACE-hardness. This leaves a gap between PSPACE and EXPTIME, which is the starting point of our work.

In this paper we show that the reachability problem for d-BoBrVASS model is EXPTIME-complete for any $d \geqslant 2$, leaving as an open problem the case of $d = 1$.

▶ **Theorem 4.** *For all $d \geqslant 2$, the reachability problem for d-BoBrVASS is EXPTIME-complete.*

The proof of Theorem 4 is split into two parts. In Section 3 we introduce auxiliary models that lie between 1-BoBrVASS and 2-BoBrVASS. Later in Section 4 we prove that these models are EXPTIME-hard.

[3] For BrVASS usually this question is phrased asking for the existence of a run not a context. In this scenario it means that in the input of the problem one requires $q(\boldsymbol{m}) = q_0(\boldsymbol{0})$. It is easy to see that the two questions are equivalent in terms of complexity for all variants of BrVASS in this paper.

3 Extensions of 1-BoBrVASS with doubling and halving

We introduce convenient extensions of the models 1-BoVASS and 1-BoBrVASS, where we add the ability of multiplying and dividing the counter value by 2. Formally, a *doubling transition* is a transition of the form $t = (p, q)_{\times 2}$ for some states p, q that has the following semantics: we can write $p(n) \xrightarrow{t} q(m)$ if and only if $m = 2n$. Similarly, a *halving transition* is a transition of the form $t = (p, q)_{\div 2}$ for some states p, q that has the following semantics: we may write $p(n) \xrightarrow{t} q(m)$ if and only if $m = n/2$. Note that in particular, if n is odd then the transition cannot be applied. Models 1-BoVASS$^{\times 2}$, 1-BoVASS$^{\div 2}$, and 1-BoVASS$^{\times 2, \div 2}$ extend 1-BoVASS by respectively allowing doubling transitions, halving transitions, and both doubling and halving transitions. Models 1-BoBrVASS$^{\div 2}$, 1-BoBrVASS$^{\times 2}$ and 1-BoBrVASS$^{\times 2, \div 2}$ extend 1-BoBrVASS in the same way.

It is not hard to see that doubling and halving transitions can be simulated using an additional counter, as explained next.

▶ **Lemma 5.** *For every 1-*BoVASS$^{\times 2, \div 2}$ $\mathcal{V} = (Q, \Delta)$ *there exists a 2-*BoVASS $\mathcal{V}' = (Q', \Delta')$ *of polynomial size in* $\mathcal{V}$ *such that* $Q \subseteq Q'$ *and* $p(n) \rightarrow^* q(m)$ *iff* $p(n, 0) \rightarrow^* q(m, 0)$*, for all* $p, q \in Q$*. Similarly, for every 1-*BoBrVASS$^{\times 2, \div 2}$ *there exists a 2-*BoBrVASS *that is analogous as above.*

Proof. Every doubling transition $(p, q)_{\times 2}$ can be simulated using the extra counter, two additional states $r_1, r_2 \notin Q$ and five transitions: $t_1 = (p, (0, 0), r_1)$, $t_2 = (r_1, (-1, 2), r_1)$, $t_3 = (r_1, r_2)_{c_1 = 0}$, $t_4 = (r_2, (1, -1), r_2)$, $t_5 = (r_2, q)_{c_2 = 0}$. Similarly, every halving transition $(p, q)_{\times 2}$ can be simulated using the extra counter, two additional states $s_1, s_2 \notin Q$ and five transitions: $t_1 = (p, (0, 0), s_1)$, $t_2 = (s_1, (-2, 1), s_1)$, $t_3 = (s_1, s_2)_{c_1 = 0}$, $t_4 = (s_2, (1, -1), s_2)$, $t_5 = (s_2, q)_{c_2 = 0}$. ◀

Therefore, 1-BoBrVASS$^{\times 2, \div 2}$ is a model in between 1-BoBrVASS and 2-BoBrVASS. This model was inspired by the *more powerful one register machine* (MP1RM) [19], used to prove the limitations of expressiveness for two-counter machines[4]. It is a one counter machine (without bounds) extended with two operations: multiplying by constants and dividing by constants. A more general model with polynomial updates was also studied in [12].

By Lemma 5, the reachability problem for d-BoBrVASS for any $d \geqslant 2$ is at least as hard as the reachability problem for 1-BoBrVASS$^{\times 2, \div 2}$. In the next section we will prove that the latter problem is already EXPTIME-hard, which will conclude the proof of Theorem 4. Before this, we show a side result which essentially says that doubling can be implemented using halving and vice versa.

▶ **Proposition 6.** *The reachability problems for models 1-*BoVASS$^{\times 2, \div 2}$*, 1-*BoVASS$^{\times 2}$ *and 1-*BoVASS$^{\div 2}$ *are equivalent with respect to polynomial time reductions. Similarly, the reachability problems for models 1-*BoBrVASS$^{\times 2, \div 2}$*, 1-*BoBrVASS$^{\times 2}$ *and 1-*BoBrVASS$^{\div 2}$ *are equivalent with respect to polynomial time reductions.*

Proof. We will give polynomial-time reductions from model 1-BoVASS$^{\times 2, \div 2}$ to models 1-BoVASS$^{\times 2}$ and 1-BoVASS$^{\div 2}$. The converse reductions are trivial, and the result for models with branching (1-BoBrVASS$^{\times 2, \div 2}$, 1-BoBrVASS$^{\times 2}$ and 1-BoBrVASS$^{\div 2}$) follows by applying the same construction.

[4] The amazing name and abbreviation of the model come from [19].

We start with the reduction from 1-BoVASS$^{\times 2,\div 2}$ to 1-BoVASS$^{\times 2}$. Consider a 1-BoVASS$^{\times 2,\div 2}$ $\mathcal{B}$, say with an upper bound B. By applying Lemma 3, we may assume that $B+1$ is a power of 2, say $B = 2^n - 1$ for some positive integer n.

The idea is to simulate every halving transition $(u, v)_{\div 2}$ in $\mathcal{B}$ with a sequence of $O(n)$ transitions (standard or doubling); since n is polynomial in the size of $\mathcal{B}$, this is fine for a polynomial-time reduction. We explain this sequence in words; translating this description into a formal definition of a system and adding appropriate states and transitions to $\mathcal{B}$ is straightforward. Apply $n-1$ times the following operation (depicted on Figure 2): if the counter value is smaller than 2^{n-1} then just double the counter; and otherwise subtract 2^{n-1} from the counter, double it and then add 1. Finally, at the end test whether the counter value is smaller than 2^{n-1}.

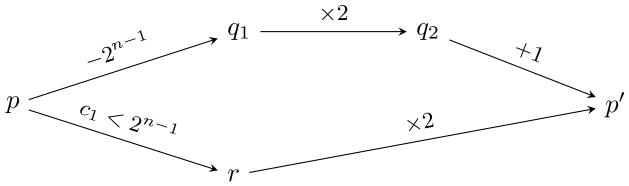


Figure 2 Shifting the bits cyclically to the left.

To see that these operations correctly implement halving the counter value, consider its bit encoding. Each single operation described above amounts to cyclically shifting the bit encoding to the left by 1: every bit is moved to a position one higher, apart from the bit from the highest (nth) position which is moved to the lowest position. Thus, after $n-1$ applications the initial bit encoding is cyclically shifted by $n-1$ to the left, which is equivalent to shifting it by 1 to the right. So now it suffices to verify that the highest bit (which was the lowest in the encoding of the initial counter value) is 0, and if this is the case, then the final counter value is equal to half of the initial counter value.

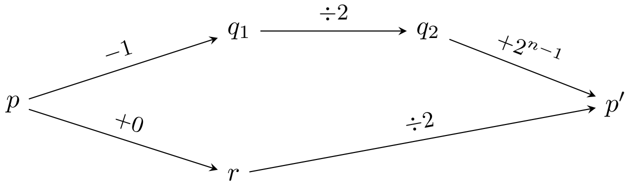


Figure 3 Shifting the bits cyclically to the right.

The reduction from 1-BoVASS$^{\times 2,\div 2}$ to 1-BoVASS$^{\div 2}$ follows by the same reasoning, but reversed. That is, we have to implement every doubling transition $(u, v)_{\times 2}$ in $\mathcal{B}$ using a sequence of $O(n)$ transitions (standard or halving). First, we verify that the counter value is smaller than 2^{n-1}, for otherwise the doubling transition cannot be applied. If this is the case, we cyclically shift the binary encoding of the counter value by 1 to the right $n-1$ times. As Figure 3 shows, each such shift is performed by first nondeterministically guessing whether the counter value is odd, subtracting 1 if this is the case, halving, and adding 2^{n-1} to the counter value if the counter value was odd. ◀

4 Exptime-hardness

This section is devoted to prove EXPTIME-completness of 1-BoBrVASS$^{\times 2,\div 2}$. Note that Theorem 7 below together with Lemma 5 immediately give Theorem 4.

▶ **Theorem 7.** *The reachability problem for 1-*BoBrVASS$^{\times 2,\div 2}$ *is EXPTIME-complete.*

We start by proving two lemmas that will be helpful in the reduction. Afterwards we define countdown games, an EXPTIME-complete problem from which we reduce. In the next two claims we write M for some power of 2, i.e. $M = 2^n$ for some $n \in \mathbb{N}$.

▶ **Lemma 8.** *Let* $f\colon \{0, 1, \ldots, M-1\} \to \{0, 1, \ldots, M-1+M(M-1)\}$ *be defined as* $f(x) = x + Mx$. *Then f is computable by a 1-*BoVASS$^{\times 2,\div 2}$ *of size polynomial in the representation of M, with an upper bound* $B = M + M(M-1) + M^2(M-1) + M^3(M-1)$.

Proof. The set of states is $Q = \{p, r, q_0, q_1, \ldots, q_n\}$. The transitions are depicted in Figure 4.

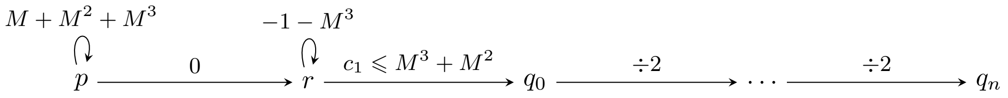


Figure 4 The 1-BoVASS$^{\times 2,\div 2}$ computing $f(x) = x + Mx$.

We now prove that the system defined above indeed computes f. Consider any $x \in \{0, 1, \ldots, M-1\}$. Observe that $p(x) \to^* r(y)$ if and only if $y = x + Ma + M^2a + M^3a - (M^3+1)b$ for some $0 \leqslant b \leqslant a \leqslant M-1$ (a is the number of loops taken in p and b is the number of loops taken in r). Then observe that $r(x + Ma + M^2a + M^3a) \to^* q_0(y)$ if and only if $y = x - a + Ma + M^2a$. Finally, note that it is possible to go from q_0 to q_n only if the initial counter value is divisible by M. Since $|x - a| < M$, this is possible only if $x = a$. Then we get $q_0(Mx + M^2x) \to^* q_n(y)$ if and only if $y = x + Mx$, so in total $p(x) \to^* q_n(y)$ if and only if $y = x + Mx$, as required. ◀

We now combine the system provided by Lemma 8 with the branching feature of the BoBrVASS model to implement the key functionality: copying the counter value into two separate branches.

▶ **Lemma 9.** *There is a 1-*BoBrVASS$^{\times 2,\div 2}$ *of size polynomial in the representation of M, with an upper bound as in Lemma 8, and distinguished states p, q_1, q_2, with the following property. For every $x \in \{0, 1, \ldots, M-1\}$ there is a unique partial run ρ_x of the system whose root is labelled with configuration $p(x)$ and whose all leaves have states q_1 or q_2. Moreover, ρ_x has exactly two leaves: one with configuration $q_1(x)$ and second with configuration $q_2(x)$.*

Proof. We start the construction with the 1-BoVASS$^{\times 2,\div 2}$ provided by Lemma 8, hence we can assume that there is a state p' such that for all $x \in \{0, 1, \ldots, M-1\}$, $p(x) \to^* p'(y)$ if and only if $y = x + Mx$. Next, add states $\{r, q_1, s_0, \ldots, s_{n-1}, q_2\}$ to the system together with the following transitions:

- branching transition (p', r, s_0);
- inequality test $(r, q_1)_{c_1 \leqslant M-1}$; and
- halving transitions $(s_i, s_{i+1})_{\div 2}$ for all $i \in \{0, 1, \ldots, n-1\}$, where $s_n = q_2$.

We now argue that this system has the required property. Observe that starting from configuration $p(x)$, the system first has to move to configuration $p'(x + Mx)$, where it branches into configurations $r(y)$ and $s_0(z)$ with $y + z = x + Mx$. In one branch we test that $y \leqslant M-1$ and end in configuration $q_1(y)$. In the other branch we divide z by 2 exactly n times, which amounts to testing that z is divisible by M and finishing in configuration $q_2(z/M)$. Since $y \leqslant M-1$, z is divisible by M, and $x \in \{0, 1, \ldots, M-1\}$, we conclude that the split (y, z) of $x + Mx$ has to be equal to (x, Mx). Hence, the only two leaves are labelled with configurations $q_1(x)$ and $q_2(x)$, respectively. ◀

We will now show a reduction from a classic EXPTIME-complete problem – determining a winner of a countdown game – to reachability in 1-BoBrVASS$^{\times 2, \div 2}$. A *countdown game* is a pair (S, T) such that S is a set of nodes partitioned into two subsets $S = S_\exists \uplus S_\forall$, and $T \subseteq S \times (\mathbb{N} \setminus \{0\}) \times S$ are weighted transitions, with weights encoded in binary. We can assume that for every node s there exists exactly two outgoing transitions (s, n, s') for some integer n and node s'. A configuration of a game is $s(c)$, where $s \in S$ and $c \in \mathbb{N}$.

The game has two players, each owning her set of nodes (Player q owns S_q, for $q \in \{\exists, \forall\}$). The game proceeds as follows: for every configuration $s(c)$ the owner of the node s chooses one of the two transitions (s, n, s') such that $n \leqslant c$ and moves to the configuration $s'(c-n)$. If it is not possible to choose such a transition, then the game ends and Player $\forall$ wins the game. Otherwise, if at some point a configuration with value 0 is reached, then the game ends and Player $\exists$ wins the game.

Given a countdown game (S, T) and a starting configuration $s(c)$ the problem whether Player $\exists$ has a winning strategy is known to be EXPTIME-complete [16][5]. Without loss of generality we can assume that $c = 2^n - 1$ for some integer n.

Proof of Theorem 7. We give a reduction from the problem of determining the winner of a countdown game. Let then (S, T) be the given countdown game and $s(c)$ be its starting configuration, where $c = 2^n - 1$ for some integer n. We are going to construct, in polynomial time, a 1-BoBrVASS$^{\times 2, \div 2}$ $\mathcal{B}$ together with a root configuration such that $\mathcal{B}$ has an accepting run from the root configuration (i.e. one where all leaves are labelled with the initial configuration $q_0(0)$) if and only if Player $\exists$ wins the game.

We start by including all nodes of S in the state set of $\mathcal{B}$. For every node $r \in S_\exists$, let (r, n_1, r'_1) and (r, n_2, r'_2) be the two transitions in T that have r as the origin. Then add transitions $(r, -n_1, r'_1)$ and $(r, -n_2, r'_2)$ to the transition set of $\mathcal{B}$.

Next, for every node $r \in S_\forall$, let (r, n_1, r'_1) and (r, n_2, r'_2) be the two transitions in T that have r as the origin. Then add a copy of the system provided by Lemma 9 for $M = c + 1$, and identify its distinguished state p with r. Denoting the other two distinguished states of the copy by q_1^r and q_2^r, add transitions $(q_1^r, -n_1, r'_1)$ and $(q_2^r, -n_2, r'_2)$ to $\mathcal{B}$.

Finally, we add an initial state q_0 together with a transition $(r, 0, q_0)$ for each $r \in S$. Thus, $\mathcal{B}$ may accept whenever it reaches any state $r \in S$ with counter value 0. This finishes the construction of $\mathcal{B}$. We set the root configuration to be $s(c)$.

To see that the reduction is correct it suffices to observe that accepting runs of $\mathcal{B}$ with root configuration $s(c)$ are in one-to-one correspondence with winning strategies of Player $\exists$ in the game (S, T) starting from $s(c)$. Indeed, in nodes belonging to Player $\exists$, she may choose any of the two transitions originating there, which corresponds to the possibility of taking any of the two transitions in an accepting run of $\mathcal{B}$. In nodes belonging to Player $\forall$, Player $\exists$ has to be able to win for both possible moves of Player $\forall$, which corresponds to requiring acceptance in both subtrees obtained as a result of running the respective copy of the system provided by Lemma 9. ◀

[5] The game considered in [16] is slightly different but it is easy to show that both definitions are equivalent.

5 Limitations of 1-BoBrVASS

The natural approach to show EXPTIME hardness for 1-BoBrVASS is to ask whether they can efficiently compute the function $f : \{0, \dots, M\} \to \{0, \dots, 2M\}$ defined by $f(x) = 2x$. Then 1-BoBrVASS could simulate 1-BoBrVASS$^{\times 2}$ and EXPTIME-hardness would follow from Theorem 7 and Proposition 6. Unfortunately, we show that this is not the case. In the following proposition, we assume the original definition of 1-BoBrVASS, without inequality or equality tests as single transitions.

▶ **Proposition 10.** *Consider the function $f \colon \{0, \dots, M\} \to \{0, \dots, 2M\}$ defined as $f(x) = 2x$, for some $M \in \mathbb{N}$. Then every 1-BoBrVASS computing the function f has at least M states.*

Proof. Suppose $\mathcal{B}$ is a 1-BoBrVASS that computes f: there are $p, q \in Q$ such that for all $x \in \{0, \dots, M\}$, $p(x) \to^* q(y)$ if and only if $y = 2x$. Here, Q denotes the set of states of $\mathcal{B}$.

For every $x \in \{0, \dots, M\}$, fix any run ρ_x that witnesses $p(x) \to^* q(2x)$. The *spine* of ρ_x is the path from the root (labelled $p(x)$) to the leaf labelled $q(2x)$ (in the special case when $q(2x) = q_0(0)$ this path might be not unique and the spine is defined as any path).

▷ Claim 11. For every $x \in \{1, \dots, M\}$, the spine of ρ_x contains at least one configuration with counter value 0.

Proof. Suppose every configuration on the spine of ρ_x has a positive counter value. Consider run ρ' obtained from ρ_x by decrementing all the counter value in each configuration on the spine by 1. Then ρ' witnesses that $p(x-1) \to^* q(2x-1)$, contradicting the assumption that $\mathcal{B}$ computes f. ◁

▷ Claim 12. For all different $x, x' \in \{0, \dots, M\}$, the sets of configurations on the spines of ρ_x and $\rho_{x'}$ are disjoint.

Proof. Suppose some configuration $r(z)$ appears both on the spine of ρ_x and of $\rho_{x'}$. Then replacing the context rooted at $r(z)$ in ρ_x with the context rooted at $r(z)$ in $\rho_{x'}$ yields a run witnessing that $p(x) \to^* q(2x')$. As $x \neq x'$, this contradicts the assumption that $\mathcal{B}$ computes f. ◁

The proposition now follows immediately from the two claims above: For every $x \in \{1, \dots, M\}$ the spine of ρ_x has to contain some configuration with counter value 0, and these configuration have to be pairwise distinct for different x. This implies that $\mathcal{B}$ has to have at least M states, as claimed. ◀

6 Conclusion

We have proved EXPTIME-completeness of reachability for 2-BoBrVASS leaving the complexity for 1-BoBrVASS between PSPACE and EXPTIME. Our EXPTIME-hardness proof for 2-BoBrVASS heavily relied on the ability of manipulating a counter by multiplying and dividing it by 2. As shown by Proposition 10, one counter alone is not enough to implement this functionality in the BoBrVASS model.

At this moment we do not have any clear indication on whether reachability for 1-BoBrVASS is EXPTIME-hard, or it rather leans towards PSPACE. On one hand, EXPTIME-hardness would give us a natural problem to reduce from for other infinite-state systems with branching, similarly to the role served by reachability for 1-BoVASS. On the other hand, PSPACE-membership would be even more interesting, as intuitively it would show that in the 1-BoBrVASS model the branching transitions are too weak to emulate alternation, contrary to the situation in multiple other models of similar kind.

To motivate the study of this problem even further, we highlight the connections between 1-BoBrVASS and 2-BrVASS. It is known that reachability for 2-VASS is PSPACE-hard, however the only proof of hardness we are aware of uses the reduction from 1-BoVASS. We show that similarly one can reduce 1-BoBrVASS to 2-BrVASS.

▶ **Proposition 13.** *The reachability problem for 1-*BoBrVASS *is reducible to the reachability problem for 2-*BrVASS *in polynomial time.*

Proof. Consider a 1-BoBrVASS $\mathcal{B}$, say with state set Q and upper bound $B \in \mathbb{N}$. The idea is to construct a 2-BrVASS $\mathcal{V}$ where every configuration $p(x)$ of $\mathcal{B}$ will be simulated by a configuration of the form $p(x, 2B - x)$ of $\mathcal{V}$.

We start by including Q in the state set of $\mathcal{V}$.

Next, we model every unary transition $t = (u, k, v)$ of $\mathcal{B}$ using two unary transitions in $\mathcal{V}$: $(u, (k, -B - k), r_t)$ followed by $(r_t, (0, B), v)$, where r_t is a new state added for the transition t. Note that this sequence of two transitions can be fired in configuration of the form $u(x, 2B - x)$ if and only if $x + k \in \{0, \ldots, B\}$, and then the resulting configuration is $v(x + k, 2B - (x + k))$.

Every branching transition $t = (u, v_1, v_2)$ of $\mathcal{B}$ is modelled in $\mathcal{V}$ as follows. We add to $\mathcal{V}$ two new states $s_{t,1}$ and $s_{t,2}$, a branching transition $(u, s_{t,1}, s_{t,2})$, and unary transitions $(s_{t,1}, (0, B), v_1)$ and $(s_{t,2}, (0, B), v_2)$. Observe that thus, after firing transition $(u, s_{t,1}, s_{t,2})$ in configuration of the form $u(x, 2B - x)$ we obtain two configurations of the form $s_{t,1}(x_1, y'_1)$ and $s_{t,2}(x_2, y'_2)$ satisfying $x_1 + x_2 = x$ and $y'_1 + y'_2 = 2B - x$, in which we have to apply the transitions that add B to the second counter, obtaining configurations $v_1(x_1, y_1)$ and $v_2(x_2, y_2)$ satisfying $x_1 + x_2 = x$ and $y_1 + y_2 = 4B - x$. Hence, provided the initial split of the second counter was $(y'_1, y'_2) = (B - x_1, B - x_2)$, we indeed obtain configurations $v_1(x_1, 2B - x_1)$ and $v_2(x_2, 2B - x_2)$, as was our initial intention. We will later argue that in runs in which we are interested, in all branching transitions the second counter has to be split as above.

Finally, if q_0 is the initial state of $\mathcal{B}$, we add a new initial state q'_0 to $\mathcal{V}$ together with a transition $(q_0, (0, -2B), q'_0)$. Thus, the only way to arrive at configuration $q'_0(0, 0)$ is to apply this transition in configuration $q_0(0, 2B)$.

Now consider any pair of configurations $p(n), q(m)$ of $\mathcal{B}$, where $n, m \in \{0, \ldots, B\}$. We claim that $p(n) \to^* q(m)$ in $\mathcal{B}$ if and only if $p(n, 2B - n) \to^* q(m, 2B - m)$ in $\mathcal{V}$.

For the forward direction, consider a partial run π of $\mathcal{B}$ witnessing $p(n) \to^* q(m)$ and modify it to a run ρ of $\mathcal{V}$ as follows. First, replace every configuration of the form $u(x)$ in π with configuration $u(x, 2B-x)$. Next, replace the transitions of $\mathcal{B}$ in π with sequences of transitions of $\mathcal{V}$ in the expected way, as described above. In particular, whenever in π a branching transition (u, v_1, v_2) is applied to configuration $u(x)$ and results in configurations $v_1(x_1)$ and $v_2(x_2)$, in ρ we model this by applying a branching transition to $u(x, 2B - x)$ and obtaining $s_{t,1}(x_1, B - x_1)$ and $s_{t,2}(x_2, B - x_2)$, which are subsequently modified to $v_1(x_1, 2B - x_1)$ and $v_2(x_2, 2B - x_2)$. Then ρ witnesses that $p(n, 2B - n) \to^* q(m, 2B - m)$ in $\mathcal{V}$.

For the backward direction, consider a partial run ρ of $\mathcal{V}$ witnessing $p(n, 2B - n) \to^* q(m, 2B - m)$. A node of ρ will be called *principal* if the state of its configuration belongs to the original state set Q; other nodes will be called *auxiliary*. Note that in ρ, every second level consists entirely of principal nodes, and every other level consists entirely of auxiliary nodes. More specifically, every principal node a, say labelled with a configuration $u(x, y)$, is of one of the following four kinds:

- a has one child and one grandchild, labelled respectively with configurations $r_t(x+k, y-k-B)$ and $v(x+k, y-k)$ for some unary transition (u, k, v) of $\mathcal{B}$;
- a has two children, each having one child of its own, and these are respectively labelled with configurations $s_{t,1}(x_1, y'_1), s_{t,2}(x_2, y'_2), v_1(x_1, y'_1+B), v_2(x_2, y'_2+B)$ for some branching transition (u, v_1, v_2) of $\mathcal{B}$;
- a has one leaf child labelled with $q'_0(0, 0)$, so in particular the configuration at a is $q(x, z) = q_0(0, 2B)$; or
- a is the unique leaf labelled with $q(m, 2B-m)$.

These types of principal nodes will be called *unary*, *branching*, *leaf*, and *final*, respectively.

We first prove by a bottom-up induction over ρ that in all principal nodes, the sum of counter values is equal to $2B$. This trivially holds for all principal nodes that are leaf or final. Next, whenever a is a unary principal node with configuration $u(x, y)$, then by the induction assumption for its grandchild we infer that $(x+k)+(y-k) = 2B$ for some integer k, implying $x+y = 2B$ as required. The case of branching principal nodes is analogous.

We now prove by a top-down induction over ρ that in all principal nodes, the value of the first counter is upper bounded by B. This is certainly true in the root, since we assumed that $n \leqslant B$. Suppose a is a unary principal node with configuration $u(x, 2B-x)$ for which we already know that $x \leqslant B$, and its child and grandchild are respectively labelled with $r_t(x+k, B-x-k)$ and $v(x+k, 2B-x-k)$ for some unary transition (u, k, v) of $\mathcal{B}$. Then the existence of the child's configuration implies that $B-x-k \geqslant 0$, implying that $x+k \leqslant B$, as required. Finally, observe that if a is a branching principal node, then the sum of the first counter values in its grandchildren is equal to the first counter value in a, hence an upper bound of B on the latter entails an upper bound of B on the former.

All in all, we argued that all principal nodes in ρ have configurations of the form $u(x, 2B-x)$, where $u \in Q$ and $x \in \{0, \ldots B\}$. It now remains to observe that taking all principal nodes with tree order induced from ρ and forgetting the second counter value yields a run of $\mathcal{B}$ witnessing that $p(n) \to^* q(m)$ in $\mathcal{B}$. ◀

References

1 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.

2 Michael Blondin, Alain Finkel, Stefan Göller, Christoph Haase, and Pierre McKenzie. Reachability in Two-Dimensional Vector Addition Systems with States Is PSPACE-Complete. In *30th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2015, Kyoto, Japan, July 6-10, 2015*, pages 32–43, 2015. doi:10.1109/LICS.2015.14.

3 Mikolaj Bojańczyk. Automata column. *SIGLOG News*, 1(2):3–12, 2014. URL: https://dl.acm.org/citation.cfm?id=2677163.

4 Mikolaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3):13:1–13:48, 2009. doi:10.1145/1516512.1516515.

5 Ahmed Bouajjani and Michael Emmi. Analysis of Recursively Parallel Programs. *ACM Trans. Program. Lang. Syst.*, 35(3):10:1–10:49, 2013. doi:10.1145/2518188.

6 Lorenzo Clemente, Sławomir Lasota, Ranko Lazi¸, and Filip Mazowiecki. Timed pushdown automata and branching vector addition systems. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005083.

7 Costas Courcoubetis and Mihalis Yannakakis. Minimum and Maximum Delay Problems in Real-Time Systems. *Formal Methods in System Design*, 1(4):385–415, 1992. doi:10.1007/BF00709157.

8 Wojciech Czerwinski and Slawomir Lasota. Regular separability of one counter automata. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12, 2017. doi:10.1109/LICS.2017.8005079.

9 Matthias Englert, Ranko Lazic, and Patrick Totzke. Reachability in Two-Dimensional Unary Vector Addition Systems with States is NL-Complete. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 477–484, 2016. doi:10.1145/2933575.2933577.

10 John Fearnley and Marcin Jurdzinski. Reachability in two-clock timed automata is PSPACE-complete. *Inf. Comput.*, 243:26–36, 2015. doi:10.1016/j.ic.2014.12.004.

11 Diego Figueira, Ranko Lazic, Jérôme Leroux, Filip Mazowiecki, and Grégoire Sutre. Polynomial-Space Completeness of Reachability for Succinct Branching VASS in Dimension One. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 119:1–119:14, 2017. doi:10.4230/LIPIcs.ICALP.2017.119.

12 Alain Finkel, Stefan Göller, and Christoph Haase. Reachability in Register Machines with Polynomial Updates. In *Mathematical Foundations of Computer Science 2013 - 38th International Symposium, MFCS 2013, Klosterneuburg, Austria, August 26-30, 2013. Proceedings*, pages 409–420, 2013. doi:10.1007/978-3-642-40313-2_37.

13 Stefan Göller, Christoph Haase, Ranko Lazic, and Patrick Totzke. A Polynomial-Time Algorithm for Reachability in Branching VASS in Dimension One. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 105:1–105:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.105.

14 Christoph Haase, Stephan Kreutzer, Joël Ouaknine, and James Worrell. Reachability in Succinct and Parametric One-Counter Automata. In *CONCUR 2009 - Concurrency Theory, 20th International Conference, CONCUR 2009, Bologna, Italy, September 1-4, 2009. Proceedings*, pages 369–383, 2009. doi:10.1007/978-3-642-04081-8_25.

15 Christoph Haase, Joël Ouaknine, and James Worrell. On the Relationship between Reachability Problems in Timed and Counter Automata. In *Reachability Problems - 6th International Workshop, RP 2012, Bordeaux, France, September 17-19, 2012. Proceedings*, pages 54–65, 2012. doi:10.1007/978-3-642-33512-9_6.

16 Marcin Jurdziński, Jeremy Sproston, and François Laroussinie. Model Checking Probabilistic Timed Automata with One or Two Clocks. *Logical Methods in Computer Science*, 4(3), 2008. doi:10.2168/LMCS-4(3:12)2008.

17 Jérôme Leroux and Sylvain Schmitz. Reachability in Vector Addition Systems is Primitive-Recursive in Fixed Dimension. In *LICS*, 2019. arXiv:1903.08575.

18 Ernst W. Mayr. An Algorithm for the General Petri Net Reachability Problem. *SIAM J. Comput.*, 13(3):441–460, 1984. doi:10.1137/0213029.

19 Rich Schroeppel. A two counter machine cannot calculate $2N$. *Artificial Intelligence Memo No. 257*, 1972.

20 Leslie G. Valiant and Mike Paterson. Deterministic One-Counter Automata. *J. Comput. Syst. Sci.*, 10(3):340–350, 1975. doi:10.1016/S0022-0000(75)80005-5.

Reasoning About Distributed Knowledge of Groups with Infinitely Many Agents

Michell Guzmán
University of Milano-Bicocca, Italy
michell.guzman@unimib.it

Sophia Knight
University of Minnesota Duluth, MN, USA
sophia.knight@gmail.com

Santiago Quintero
LIX École Polytechnique de Paris, France
squinter@lix.polytechnique.fr

Sergio Ramírez
Pontificia Universidad Javeriana de Cali, Colombia
sergio@javerianacali.edu.co

Camilo Rueda
Pontificia Universidad Javeriana de Cali, Colombia
crueda@javerianacali.edu.co

Frank Valencia
CNRS, LIX École Polytechnique de Paris, France
Pontificia Universidad Javeriana de Cali, Colombia
frank.valencia@lix.polytechnique.fr

Abstract

Spatial constraint systems (scs) are semantic structures for reasoning about spatial and epistemic information in concurrent systems. We develop the theory of scs to reason about the *distributed information* of potentially *infinite groups*. We characterize the notion of distributed information of a group of agents as the infimum of the set of join-preserving functions that represent the spaces of the agents in the group. We provide an alternative characterization of this notion as the greatest family of join-preserving functions that satisfy certain basic properties. We show compositionality results for these characterizations and conditions under which information that can be obtained by an infinite group can also be obtained by a finite group. Finally, we provide algorithms that compute the distributive group information of finite groups.

2012 ACM Subject Classification Theory of computation → Concurrency; Theory of computation → Distributed computing models; Theory of computation → Semantics and reasoning

Keywords and phrases Reasoning about Groups, Distributed Knowledge, Infinitely Many Agents, Reasoning about Space, Algebraic Modeling

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.29

Related Version Full version of this paper at `https://hal.archives-ouvertes.fr/hal-02172415`.

Funding This work have been partially supported by the ECOS-NORD project FACTS (C19M03) and the Colciencias project CLASSIC (125171250031).

1 Introduction

In current distributed systems such as social networks, actors behave more as members of a certain *group* than as isolated individuals. Information, opinions, and beliefs of a particular actor are frequently the result of an evolving process of interchanges with other actors in a

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 29; pp. 29:1–29:15
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

group. This suggests a reified notion of group as a single actor operating within the context of the collective information of its members. It also conveys two notions of information, one spatial and the other epistemic. In the former, information is localized in compartments associated with a user or group. In the latter, it refers to something known or believed by a single agent or collectively by a group.

In this paper we pursue the development of a principled account of a reified notion of group by taking inspiration from the epistemic notion of *distributed knowledge* [12]. A group has its information distributed among its member agents. We thus develop a theory about what exactly is the information available to agents as a group when considering all that is distributed among its members.

In our account a group acts itself as an agent carrying the collective information of its members. We can interrogate, for instance, whether there is a potential contradiction or unwanted distributed information that a group might be involved in among its members or by integrating a certain agent. This is a fundamental question since it may predict or prevent potentially dangerous evolutions of the system.

Furthermore, in many real life multi-agent systems, the agents are unknown in advance. New agents can subscribe to the system in unpredictable ways. Thus, there is usually no a-priori bound on the number of agents in the system. It is then often convenient to model the group of agents as an infinite set. In fact, in models from economics and epistemic logic [14, 13], groups of agents have been represented as infinite, even uncountable, sets. In accordance with this fact, in this paper we consider that groups of agents can also be infinite. This raises interesting issues about the distributed information of such groups. In particular, that of *group compactness*: information that when obtained by an infinite group can also be obtained by one of its finite subgroups. We will provide conditions for this to hold.

Context. Constraint systems (cs)[1] are algebraic structures for the semantics of process calculi from concurrent constraint programming (ccp) [18]. In this paper we shall study cs as semantic structures for distributed information of a group of agents.

A cs can be formalized as a complete lattice $(Con, \sqsubseteq)$. The elements of Con represent partial information and we shall think of them as being *assertions*. They are traditionally referred to as *constraints* since they naturally express partial information (e.g., $x > 42$). The order $\sqsubseteq$ corresponds to entailment between constraints, $c \sqsubseteq d$, often written $d \sqsupseteq c$, means c can be derived from d, or that d represents as much information as c. The join $\sqcup$, the bottom *true*, and the top *false* of the lattice correspond to conjunction, the empty information, and the join of all (possibly inconsistent) information, respectively.

The notion of computational space and the epistemic notion of belief in the spatial ccp (sccp) process calculi [15] is represented as a family of join-preserving maps $\mathfrak{s}_i : Con \to Con$ called *space functions*. A cs equipped with space functions is called a *spatial constraint system* (scs). From a *computational point of view* $\mathfrak{s}_i(c)$ can be interpreted as an assertion specifying that c resides within the space of agent i. From an *epistemic point of view*, $\mathfrak{s}_i(c)$ specifies that i considers c to be true. An alternative epistemic view is that i interprets c as $\mathfrak{s}_i(c)$. All these interpretations convey the idea of c being local or subjective to agent i.

This work. In the spatial ccp process calculus *sccp* [15], scs are used to specify the spatial distribution of information in configurations $\langle P, c\rangle$ where P is a process and c is a constraint, called *the store*, representing the current partial information. E.g., a reduction $\langle\ P, \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b)\ \rangle \longrightarrow \langle\ Q, \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b \sqcup c)\ \rangle$ means that P, with a in the space of agent 1 and b in the space of agent 2, can evolve to Q while adding c to the space of agent 2.

[1] For simplicity we use *cs* for both *constraint system* and its plural form.

Given the above reduction, assume that d is some piece of information resulting from the combination (join) of the three constraints above, i.e., $d = a \sqcup b \sqcup c$, but strictly above the join of any two of them. We are then in the situation where neither agent has d in their spaces, but as a group they could potentially have d by combining their information. Intuitively, d is distributed in the spaces of the group $I = \{1, 2\}$. Being able to predict the information that agents 1 and 2 may derive as group is a relevant issue in multi-agent concurrent systems, particularly if d represents unwanted or conflicting information (e.g., $d = \mathit{false}$).

In this work we introduce the theory of group space functions $\Delta_I : \mathit{Con} \to \mathit{Con}$ to reason about information distributed among the members of a potentially infinite group I. We shall refer to Δ_I as the *distributed space* of group I. In our theory $c \sqsupseteq \Delta_I(e)$ holds exactly when we can derive from c that e is distributed among the agents in I. E.g., for d above, we should have $\mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b \sqcup c) \sqsupseteq \Delta_{\{1,2\}}(d)$ meaning that from the information $\mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(b \sqcup c)$ we can derive that d is distributed among the group $I = \{1, 2\}$. Furthermore, $\Delta_I(e) \sqsupseteq \Delta_J(e)$ holds whenever $I \subseteq J$ since if e is distributed among a group I, it should also be distributed in a group that includes the agents of I.

Distributed information of infinite groups can be used to reason about multi-agent computations with unboundedly many agents. For example, a *computation* in sccp is a possibly infinite reduction sequence γ of the form $\langle\ P_0, c_0\ \rangle \longrightarrow \langle\ P_1, c_1\ \rangle \longrightarrow \cdots$ with $c_0 \sqsubseteq c_1 \sqsubseteq \cdots$. The *result* of γ is $\bigsqcup_{n \geq 0} c_n$, the join of all the stores in the computation. In sccp all fair computations from a configuration have the same result [15]. Thus, the *observable behaviour* of P with initial store c, written $\mathcal{O}(P, c)$, is defined as the result of any fair computation starting from $\langle P, c\rangle$. Now consider a setting where in addition to their sccp capabilities in [15], processes can also create new agents. Hence, unboundedly many agents, say agents $1, 2, \ldots$, may be created during an infinite computation. In this case, $\mathcal{O}(P, c) \sqsupseteq \Delta_{\mathbb{N}}(\mathit{false})$, where $\mathbb{N}$ is the set of natural numbers, would imply that some (finite or infinite) set of agents in any fair computation from $\langle P, c\rangle$ may reach contradictory local information among them. Notice that from the above-mentioned properties of distributed spaces, the existence of a finite set of agents $H \subseteq \mathbb{N}$ such that $\mathcal{O}(P, c) \sqsupseteq \Delta_H(\mathit{false})$ implies $\mathcal{O}(P, c) \sqsupseteq \Delta_{\mathbb{N}}(\mathit{false})$. The converse of this implication will be called *group compactness* and we will provide meaningful sufficient conditions for it to hold.

Our main contributions are listed below.

1. We characterize the distributed space Δ_I as a space function resulting from the infimum of the set of join-preserving functions that represent the spaces of the agents of a *possibly infinite* group I.
2. We provide an alternative characterization of a distributed space as the greatest join preserving function that satisfies certain basic properties.
3. We show that distributed spaces have an inherent *compositional* nature: The information of a group is determined by that of its subgroups.
4. We provide a *group compactness* result for groups: Given an infinite group I, meaningful conditions under which $c \sqsupseteq \Delta_I(e)$ implies $c \sqsupseteq \Delta_J(e)$ for some finite group $J \subseteq I$.
5. For finite scs we shall provide *algorithms* to compute Δ_I that exploit the above-mentioned compositional nature of distributed spaces.

All in all, in this paper we put forward an algebraic theory for group reasoning in the context of ccp. The theory and algorithms here developed can be used in the semantics of the spatial ccp process calculus to reason about or prevent potential unwanted evolutions of ccp processes. One could imagine the incorporation of group reasoning in a variety of process algebraic settings and indeed we expect that such formalisms will appear in due course.

2 Background

We presuppose basic knowledge of domain and order theory [3, 1, 6] and use the following notions. Let $\mathbf{C}$ be a poset $(Con, \sqsubseteq)$, and let $S \subseteq Con$. We use $\bigsqcup S$ to denote the least upper bound (or *supremum* or *join*) of the elements in S, and $\bigsqcap S$ is the greatest lower bound (glb) (*infimum* or *meet*) of the elements in S. An element $e \in S$ is the *greatest element* of S iff for every element $e' \in S$, $e' \sqsubseteq e$. If such e exists, we denote it by $max\ S$. As usual, if $S = \{c, d\}$, $c \sqcup d$ and $c \sqcap d$ represent $\bigsqcup S$ and $\bigsqcap S$, respectively. If $S = \emptyset$, we denote $\bigsqcup S = true$ and $\bigsqcap S = false$. We say that $\mathbf{C}$ is a *complete lattice* iff each subset of Con has a supremum in Con. The poset $\mathbf{C}$ is *distributive* iff for every $a, b, c \in Con$, $a \sqcup (b \sqcap c) = (a \sqcup b) \sqcap (a \sqcup c)$. A non-empty set $S \subseteq Con$ is *directed* iff for every pair of elements $x, y \in S$, there exists $z \in S$ such that $x \sqsubseteq z$ and $y \sqsubseteq z$, or iff every *finite* subset of S has an upper bound in S. Also $c \in Con$ is *compact* iff for any directed subset D of Con, $c \sqsubseteq \bigsqcup D$ implies $c \sqsubseteq d$ for some $d \in D$. A *self-map* on Con is a function f from Con to Con. Let $(Con, \sqsubseteq)$ be a complete lattice. The self-map f on Con *preserves* the join of a set $S \subseteq Con$ iff $f(\bigsqcup S) = \bigsqcup\{f(c) \mid c \in S\}$. A self-map that preserves the join of finite sets is called *join-homomorphism*. A self-map f on Con is *monotonic* if $a \sqsubseteq b$ implies $f(a) \sqsubseteq f(b)$. We say that f *distributes* over joins (or that f *preserves* joins) iff it preserves the join of arbitrary sets. A self-map f on Con is *continuous* iff it preserves the join of any directed set.

3 Spatial Constraint Systems

Constraint systems [18] are semantic structures to specify partial information. They can be formalized as complete lattices [2].

▶ **Definition 1** (Constraint Systems [2]). *A* constraint system *(cs)* $\mathbf{C}$ *is a complete lattice* $(Con, \sqsubseteq)$. *The elements of* Con *are called* constraints. *The symbols* $\sqcup$, *true and false will be used to denote the least upper bound (lub) operation, the bottom, and the top element of* $\mathbf{C}$.

The elements of the lattice, the *constraints*, represent (partial) information. A constraint c can be viewed as an *assertion*. The lattice order $\sqsubseteq$ is meant to capture entailment of information: $c \sqsubseteq d$, alternatively written $d \sqsupseteq c$, means that the assertion d represents at least as much information as c. We think of $d \sqsupseteq c$ as saying that d *entails* c or that c can be *derived* from d. The operator $\sqcup$ represents join of information; $c \sqcup d$ can be seen as an assertion stating that both c and d hold. We can think of $\sqcup$ as representing conjunction of assertions. The top element represents the join of all, possibly inconsistent, information, hence it is referred to as *false*. The bottom element *true* represents *empty information.* We say that c is *consistent* if $c \neq false$, otherwise we say that c is *inconsistent.* Similarly, we say that c is consistent/inconsistent with d if $c \sqcup d$ is consistent/inconsistent.

Constraint Frames. One can define a general form of implication by adapting the corresponding notion from Heyting Algebras to cs. A *Heyting implication* $c \rightarrow d$ in our setting corresponds to the *weakest constraint* one needs to join c with to derive d.

▶ **Definition 2** (Constraint Frames [7]). *A constraint system* $(Con, \sqsubseteq)$ *is said to be a* constraint frame *iff its joins distribute over arbitrary meets. More precisely,* $c \sqcup \bigsqcap S = \bigsqcap\{c \sqcup e \mid e \in S\}$ *for every* $c \in Con$ *and* $S \subseteq Con$. *Define* $c \rightarrow d$ *as* $\bigsqcap\{e \in Con \mid c \sqcup e \sqsupseteq d\}$.

The following properties of Heyting implication correspond to standard logical properties (with $\rightarrow$, $\sqcup$, and $\sqsupseteq$ interpreted as implication, conjunction, and entailment).

▶ **Proposition 3** ([7]). *Let $(Con, \sqsubseteq)$ be a constraint frame. For every $c, d, e \in Con$ the following holds: (1) $c \sqcup (c \rightarrow d) = c \sqcup d$, (2) $(c \rightarrow d) \sqsubseteq d$, (3) $c \rightarrow d = true$ iff $c \sqsupseteq d$.*

Spatial Constraint Systems. The authors of [15] extended the notion of cs to account for distributed and multi-agent scenarios with a finite number of agents, each having their own space for local information and their computations. The extended structures are called spatial cs (scs). Here we adapt scs to reason about possibly infinite groups of agents.

A *group* G is a set of agents. Each $i \in G$ has a *space* function $\mathfrak{s}_i : Con \rightarrow Con$ satisfying some structural conditions. Recall that constraints can be viewed as assertions. Thus given $c \in Con$, we can then think of the constraint $\mathfrak{s}_i(c)$ as an assertion stating that c is a piece of information residing *within a space of agent* i. Some alternative *epistemic* interpretations of $\mathfrak{s}_i(c)$ is that it is an assertion stating that agent i *believes* c, that c holds within the space of agent i, or that agent i *interprets* c as $\mathfrak{s}_i(c)$. All these interpretations convey the idea that c is local or subjective to agent i.

In [15] scs are used to specify the spatial distribution of information in configurations $\langle P, c\rangle$ where P is a process and c is a constraint. E.g., a reduction $\langle\ P, \mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d)\ \rangle \longrightarrow \langle\ Q, \mathfrak{s}_i(c) \sqcup \mathfrak{s}_j(d \sqcup e)\ \rangle$ means that P with c in the space of agent i and d in the space of agent j can evolve to Q while adding e to the space of agent j.

We now introduce the notion of space function.

▶ **Definition 4** (Space Functions). *A* space function *over a cs $(Con, \sqsubseteq)$ is a* continuous *self-map $f : Con \rightarrow Con$ s.t. for every $c, d \in Con$ (S.1) $f(true) = true$, (S.2) $f(c \sqcup d) = f(c) \sqcup f(d)$. We shall use $\mathcal{S}(\mathbf{C})$ to denote the set of all space functions over $\mathbf{C} = (Con, \sqsubseteq)$.*

The assertion $f(c)$ can be viewed as saying that c is in the space represented by f. Property S.1 states that having an empty local space amounts to nothing. Property S.2 allows us to join and distribute the information in the space represented by f.

In [15] space functions were not required to be continuous. Nevertheless, we will argue later, in Remark 17, that continuity comes naturally in the intended phenomena we wish to capture: modelling information of possibly *infinite* groups. In fact, in [15] scs could only have finitely many agents.

In this work we also extend scs to allow arbitrary, possibly infinite, sets of agents. A *spatial cs* is a cs with a possibly infinite group of agents each having a space function.

▶ **Definition 5** (Spatial Constraint Systems). *A* spatial cs *(scs) is a cs $\mathbf{C} = (Con, \sqsubseteq)$ equipped with a possibly infinite tuple $\mathfrak{s} = (\mathfrak{s}_i)_{i \in G}$ of space functions from $\mathcal{S}(\mathbf{C})$.*

We shall use $(Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ to denote an scs with a tuple $(\mathfrak{s}_i)_{i \in G}$. We refer to G and $\mathfrak{s}$ as the group of agents *and* space tuple *of $\mathbf{C}$ and to each $\mathfrak{s}_i$ as the* space function *in $\mathbf{C}$ of agent i. Subsets of G are also referred to as groups of agents (or sub-groups of G).*

Let us illustrate a simple scs that will be used throughout the paper.

▶ **Example 6.** The scs $(Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in \{1,2\}})$ in Fig. 1 is given by the complete lattice $\mathbf{M}_2$ and two agents. We have $Con = \{p \vee \neg p, p, \neg p, p \wedge \neg p\}$ and $c \sqsubseteq d$ iff c is a logical consequence of d. The top element *false* is $p \wedge \neg p$, the bottom element *true* is $p \vee \neg p$, and the constraints p and $\neg p$ are incomparable with each other. The set of agents is $\{1, 2\}$ with space functions $\mathfrak{s}_1$ and $\mathfrak{s}_2$: For agent 1, $\mathfrak{s}_1(p) = \neg p$, $\mathfrak{s}_1(\neg p) = p$, $\mathfrak{s}_1(\mathit{false}) = \mathit{false}$, $\mathfrak{s}_1(\mathit{true}) = \mathit{true}$, and for agent 2, $\mathfrak{s}_2(p) = \mathit{false} = \mathfrak{s}_2(\mathit{false})$, $\mathfrak{s}_2(\neg p) = \neg p$, $\mathfrak{s}_2(\mathit{true}) = \mathit{true}$. The intuition is that the agent 2 sees no difference between p and *false* while agent 1 interprets $\neg p$ as p and vice versa.

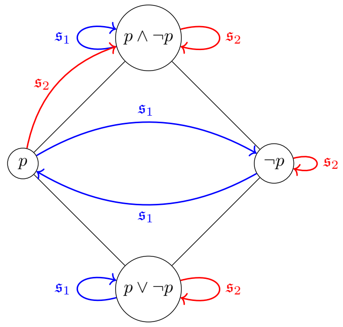


Figure 1 Cs given by lattice $\mathbf{M}_2$ ordered by implication and space functions $\mathfrak{s}_1$ and $\mathfrak{s}_2$.

More involved examples of scs include meaningful families of structures from logic and economics such as Kripke structures and Aumann structures (see [15]). We illustrate scs with infinite groups in the next section.

4 Distributed Information

In this section we characterize the notion of collective information of a group of agents. Roughly speaking, the *distributed (or collective) information* of a group I is the join of each piece of information that resides in the space of *some* $i \in I$. The distributed information of I w.r.t. c is the distributive information of I that can be derived from c. We wish to formalize whether a given e can be derived from the collective information of the group I w.r.t. c.

The following examples, which we will use throughout this section, illustrate the above intuition.

▶ **Example 7.** Consider an scs $(Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ where $G = \mathbb{N}$ and $(Con, \sqsubseteq)$ is a constraint frame. Let $c \stackrel{\texttt{def}}{=} \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(a \to b) \sqcup \mathfrak{s}_3(b \to e)$. The constraint c specifies the situation where $a, a \to b$ and $b \to e$ are in the spaces of agent 1, 2 and 3, respectively. Neither agent necessarily holds e in their space in c. Nevertheless, the information e can be derived from the collective information of the three agents w.r.t. c, since from Prop. 3 we have $a \sqcup (a \to b) \sqcup (b \to e) \sqsupseteq e$. Let us now consider an example with infinitely many agents. Let $c' \stackrel{\texttt{def}}{=} \bigsqcup_{i \in \mathbb{N}} \mathfrak{s}_i(a_i)$ for some increasing chain $a_0 \sqsubseteq a_1 \sqsubseteq \ldots$. Take e' s.t. $e' \sqsubseteq \bigsqcup_{i \in \mathbb{N}} a_i$. Notice that unless e' is compact (see Section 2), it may be the case that no agent $i \in \mathbb{N}$ holds e' in their space; e.g., if $e' \sqsupset a_i$ for any $i \in \mathbb{N}$. Yet, from our assumption, e' can be derived from the collective information w.r.t. c' of all the agents in $\mathbb{N}$, i.e., $\bigsqcup_{i \in \mathbb{N}} a_i$.

The above example may suggest that the distributed information can be obtained by joining individual local information derived from c. Individual information of an agent i can be characterized as the i-projection of c defined thus:

▶ **Definition 8** (Agent and Join Projections). *Let $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ be an scs. Given $i \in G$, the i-agent projection of $c \in Con$ is defined as $\pi_i(c) \stackrel{\texttt{def}}{=} \bigsqcup\{e \mid c \sqsupseteq \mathfrak{s}_i(e)\}$. We say that e is i-agent derivable from c iff $\pi_i(c) \sqsupseteq e$. Given $I \subseteq G$ the I-join projection of a group I of c is defined as $\pi_I(c) \stackrel{\texttt{def}}{=} \bigsqcup\{\pi_i(c) \mid i \in I\}$. We say that e is I-join derivable from c iff $\pi_I(c) \sqsupseteq e$.*

The i-projection of an agent i of c naturally represents the join of all the information of agent i in c. The I-join projection of group I joins individual i-projections of c for $i \in I$. This projection can be used as a sound mechanism for reasoning about distributed-information: If e is I-join derivable from c then it follows from the distributed-information of I w.r.t. c.

▶ **Example 9.** Let c be as in Ex. 7. We have $\pi_1(c) \sqsupseteq a$, $\pi_2(c) \sqsupseteq (a \to b)$, $\pi_3(c) \sqsupseteq (b \to e)$. Indeed e is I-join derivable from c since $\pi_{\{1,2,3\}}(c) = \pi_1(c) \sqcup \pi_2(c) \sqcup \pi_3(c) \sqsupseteq e$. Similarly we conclude that e' is I-join derivable from c' in Ex. 7 since $\pi_{\mathbb{N}}(c') = \bigsqcup_{i\in\mathbb{N}} \pi_i(c) \sqsupseteq \bigsqcup_{i\in\mathbb{N}} a_i \sqsupseteq e'$.

Nevertheless, I-join projections do not provide a complete mechanism for reasoning about distributed information as illustrated below.

▶ **Example 10.** Let $d \stackrel{\text{def}}{=} \mathfrak{s}_1(b) \sqcap \mathfrak{s}_2(b)$. Recall that we think of $\sqcup$ and $\sqcap$ as conjunction and disjunction of assertions: d specifies that b is present in the space of agent 1 or in the space of agent 2 though not exactly in which one. Thus from d we should be able to conclude that b belongs to the space of *some* agent in $\{1, 2\}$. Nevertheless, in general b is not I-join derivable from d since from $\pi_{\{1,2\}}(d) = \pi_1(d) \sqcup \pi_2(d)$ we cannot, in general, derive b. To see this consider the scs in Fig. 2a and take $b = \neg p$. We have $\pi_{\{1,2\}}(d) = \pi_1(d) \sqcup \pi_2(d) = \mathit{true} \sqcup \mathit{true} = \mathit{true} \not\sqsupseteq b$. One can generalize the example to infinitely many agents: Consider the scs in Ex. 7. Let $d' \stackrel{\text{def}}{=} \bigsqcap_{i\in\mathbb{N}} \mathfrak{s}_i(b')$. We should be able to conclude from d' that b' is in the space of *some* agent in $\mathbb{N}$ but, in general, b' is not $\mathbb{N}$-join derivable from d'.

4.1 Distributed Spaces

In the previous section we illustrated that the I-join projection of c, $\pi_I(c)$, the join of individual projections, may not project all distributed information of a group I. To solve this problem we shall develop the notion of I-group projection of c, written as $\Pi_I(c)$. To do this we shall first define a space function Δ_I called the distributed space of group I. The function Δ_I can be thought of as a virtual space including all the information that can be in the space of a member of I. We shall then define an I-projection Π_I in terms of Δ_I much like π_i is defined in terms of $\mathfrak{s}_i$.

Recall that $\mathcal{S}(\mathbf{C})$ denotes the set of all space functions over a cs $\mathbf{C}$. For notational convenience, we shall use $(f_I)_{I\subseteq G}$ to denote the tuple $(f_I)_{I\in\mathcal{P}(G)}$ of elements of $\mathcal{S}(\mathbf{C})$.

Set of Space Functions. We begin by introducing a new partial order induced by $\mathbf{C}$. The set of space functions ordered point-wise.

▶ **Definition 11** (Space Functions Order). *Let $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ be an scs. Given $f, g \in \mathcal{S}(\mathbf{C})$, define $f \sqsubseteq_s g$ iff $f(c) \sqsubseteq g(c)$ for every $c \in Con$. We shall use $\mathbf{C}_s$ to denote the partial order $(\mathcal{S}(\mathbf{C}), \sqsubseteq_s)$; the set of all space functions ordered by $\sqsubseteq_s$.*

A very important fact for the design of our structure is that the set of space functions $\mathcal{S}(\mathbf{C})$ can be made into a complete lattice.

▶ **Lemma 12.** *Let $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ be an scs. Then $\mathbf{C}_s$ is a complete lattice.*

4.2 Distributed Spaces as Maximum Spaces

Let us consider the lattice of space functions $\mathbf{C}_s = (\mathcal{S}(\mathbf{C}), \sqsubseteq_s)$. Suppose that f and g are space functions in $\mathbf{C}_s$ with $f \sqsubseteq_s g$. Intuitively, every piece of information c in the space represented by g is also in the space represented by f since $f(c) \sqsubseteq g(c)$ for every $c \in Con$. This can be interpreted as saying that the space represented by g is included in the space represented by f; in other words the bigger the space, the smaller the function that represents it in the lattice $\mathbf{C}_s$.

Following the above intuition, the order relation $\sqsubseteq_s$ of $\mathbf{C}_s$ represents (reverse) space inclusion and the join and meet operations in $\mathbf{C}_s$ represent intersection and union of spaces. The biggest and the smallest spaces are represented by the bottom and the top elements of the lattice $\mathbf{C}_s$, here called $\lambda_\bot$ and $\lambda_\top$ and defined as follows.

▶ **Definition 13** (Top and Bottom Spaces). *For every $c \in$ Con, define $\lambda_\perp(c) \overset{\text{def}}{=}$ true, $\lambda_\top(c) \overset{\text{def}}{=}$ true if $c =$ true and $\lambda_\top(c) \overset{\text{def}}{=}$ false if $c \neq$ true.*

The distributed space Δ_I of a group I can be viewed as the function that represents the smallest space that includes all the local information of the agents in I. From the above intuition, Δ_I should be the *greatest space function* below the space functions of the agents in I. The existence of such a function follows from completeness of $(\mathcal{S}(\mathbf{C}), \sqsubseteq_{\mathtt{s}})$ (Lemma 12).

▶ **Definition 14** (Distributed Spaces). *Let* $\mathbf{C}$ *be an scs* $(Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$*. The* distributed spaces *of* $\mathbf{C}$ *is given by* $\Delta = (\Delta_I)_{I \subseteq G}$ *where* $\Delta_I \overset{\text{def}}{=} max\{f \in \mathcal{S}(\mathbf{C}) \mid f \sqsubseteq_s \mathfrak{s}_i \text{ for every } i \in I\}$*. We shall say that e is distributed among* $I \subseteq G$ *w.r.t. c iff* $c \sqsupseteq \Delta_I(e)$*. We shall refer to each* Δ_I *as the* (distributed) space *of the group* I*.*

It follows from Lemma 12 that $\Delta_I = \bigsqcap\{\mathfrak{s}_i \mid i \in I\}$ (where $\bigsqcap$ is the meet in the complete lattice $(\mathcal{S}(\mathbf{C}), \sqsubseteq_{\mathtt{s}})$). Fig. 2b illustrates an scs and its distributed space $\Delta_{\{1,2\}}$.

Compositionality . Distributed spaces have pleasant compositional properties. They capture the intuition that the *distributed information* of a group I can be obtained from the the distributive information of its subgroups.

▶ **Theorem 15.** *Let* $(\Delta_I)_{I \subseteq G}$ *be the distributed spaces of an scs* $(Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$*. Suppose that* $K, J \subseteq I \subseteq G$*. (1)* $\Delta_I = \lambda_\top$ *if* $I = \emptyset$*, (2)* $\Delta_I = \mathfrak{s}_i$ *if* $I = \{i\}$*, (3)* $\Delta_J(a) \sqcup \Delta_K(b) \sqsupseteq \Delta_I(a \sqcup b)$*, and (4)* $\Delta_J(a) \sqcup \Delta_K(a \to c) \sqsupseteq \Delta_I(c)$ *if* $(Con, \sqsubseteq)$ *is a constraint frame.*

Recall that $\lambda_\top$ corresponds to the empty space (see Def. 13). The first property realizes the intuition that the empty subgroup $\emptyset$ *does not* have any information whatsoever distributed w.r.t. a consistent c: for if $c \sqsupseteq \Delta_\emptyset(e)$ and $c \neq$ *false* then $e =$ *true*. Intuitively, the second property says that the function Δ_I for the group of one agent must be the agent's space function. The third property states that a group can join the information of its subgroups. The last property uses constraint implication, hence the constraint frame condition, to express that by joining the information a and $a \to c$ of their subgroups, the group I can obtain c.

Let us illustrate how to derive information of a group from smaller ones using Thm. 15.

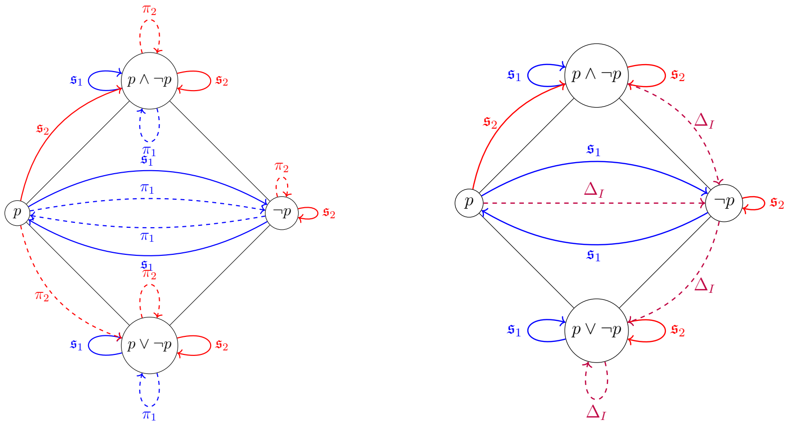
▶ **Example 16.** Let $c = \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(a \to b) \sqcup \mathfrak{s}_3(b \to e)$ as in Ex. 7. We want to prove that e is distributed among $I = \{1, 2, 3\}$ w.r.t. c, i.e., $c \sqsupseteq \Delta_{\{1,2,3\}}(e)$. Using Properties 2 and 4 in Thm. 15 we obtain $c \sqsupseteq \mathfrak{s}_1(a) \sqcup \mathfrak{s}_2(a \to b) = \Delta_{\{1\}}(a) \sqcup \Delta_{\{2\}}(a \to b) \sqsupseteq \Delta_{\{1,2\}}(b)$, and then $c \sqsupseteq \Delta_{\{1,2\}}(b) \sqcup \mathfrak{s}_3(b \to e) = \Delta_{\{1,2\}}(b) \sqcup \Delta_{\{3\}}(b \to e) \sqsupseteq \Delta_{\{1,2,3\}}(e)$ as wanted.

▶ Remark 17 (Continuity). The example with infinitely many agents in Ex. 7 illustrates well why we require our spaces to be continuous in the presence of possibly infinite groups. Clearly $c' = \bigsqcup_{i \in \mathbb{N}} \mathfrak{s}_i(a_i) \sqsupseteq \bigsqcup_{i \in \mathbb{N}} \Delta_{\mathbb{N}}(a_i)$. By continuity, $\bigsqcup_{i \in \mathbb{N}} \Delta_{\mathbb{N}}(a_i) = \Delta_{\mathbb{N}}(\bigsqcup_{i \in \mathbb{N}} a_i)$ which indeed captures the idea that each a_i is in the distributed space $\Delta_{\mathbb{N}}$.

In Thm. 15 we listed some useful properties about $(\Delta_I)_{I \subseteq G}$. In the next section we shall see that $(\Delta_I)_{I \subseteq G}$ is the greatest solution of three basic properties.

We conclude this subsection with an important family of scs from mathematical economics: Aumann structures. We illustrate that the notion of distributed knowledge in these structures is an instance of a distributed space.

▶ **Example 18** (Aumann Constraint Systems). Aumann structures [13] are an *event-based* approach to modelling knowledge. An Aumann structure is a tuple $\mathcal{A} = (S, \mathcal{P}_1, \ldots, \mathcal{P}_n)$ where S is a set of states and each $\mathcal{P}_i$ is a partition on S for agent i. The partitions are



(a) Projections π_1 and π_2 given $\mathfrak{s}_1$ and $\mathfrak{s}_2$. **(b)** Δ_I with $I = \{1, 2\}$ given $\mathfrak{s}_1$ and $\mathfrak{s}_2$.

Figure 2 Projections (a) and Distributed Space function (b) over lattice $\mathbf{M}_2$.

called *information sets*. If two states t and u are in the same information set for agent i, it means that in state t agent i considers state u possible, and vice versa. An *event* in an Aumann structure is any subset of S. Event e holds at state t if $t \in e$. The set $\mathcal{P}_i(s)$ denotes the information set of $\mathcal{P}_i$ containing s. The event of *agent i knowing e* is defined as $\mathsf{K}_i(e) = \{s \in S \mid \mathcal{P}_i(s) \subseteq e\}$, and the *distributed knowledge of an event e among the agents in a group I* is defined as $\mathsf{D}_I(e) = \{s \in S \mid \bigcap_{i \in I} \mathcal{P}_i(s) \subseteq e\}$.

An Aumann structure can be seen as a spatial constraint system $\mathbf{C}(\mathcal{A})$ with events as constraints, i.e., $Con = \{e \mid e \text{ is an event in } \mathcal{A}\}$, and for every $e_1, e_2 \in Con$, $e_1 \sqsubseteq e_2$ iff $e_2 \subseteq e_1$. The operators join ($\sqcup$) and meet ($\sqcap$) are intersection ($\cap$) and union ($\cup$) of events, respectively; *true* $= S$ and *false* $= \emptyset$. The space functions are the knowledge operators, i.e., $\mathfrak{s}_i(c) = \mathsf{K}_i(c)$. From these definitions and since meets are unions one can easily verify that $\Delta_I(c) = \mathsf{D}_I(c)$ which shows the correspondence between distributed information and distributed knowledge.

4.3 Distributed Spaces as Group Distributions Candidates

We now wish to single out a few fundamental properties on tuples of self-maps that can be used to characterize distributed spaces.

▶ **Definition 19** (Distribution Candidates). *Let $\mathbf{C}$ be an scs $(Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$. A tuple $\delta = (\delta_I)_{I \subseteq G}$ of self-maps on Con is a* group distribution candidate *(gdc) of $\mathbf{C}$ if for each $I, J \subseteq G$: (D.1) δ_I is a space function in $\mathbf{C}$, (D.2) $\delta_I = \mathfrak{s}_i$ if $I = \{i\}$, (D.3) $\delta_I \sqsupseteq_s \delta_J$ if $I \subseteq J$.*

Property D.1 requires each δ_I to be a space function. This is trivially met for $\delta_I = \Delta_I$. Property D.2 says that the function δ_I for a group of one agent must be the agent's space function. Clearly, $\delta_{\{i\}} = \Delta_{\{i\}}$ satisfies D.2; indeed the distributed space of a single agent is their own space. Finally, Property D.3 states that $\delta_I(c) \sqsupseteq \delta_J(c)$, if $I \subseteq J$. This is also trivially satisfied if we take $\delta_I = \Delta_I$ and $\delta_J = \Delta_J$. Indeed if a subgroup I has some distributed information c then any subgroup J that includes I should also have c. This also realizes our intuition above: The bigger the group, the bigger the space and thus the smaller the space function that represents it.

Properties D1–D3, however, do not determine Δ uniquely. In fact, there could be infinitely-many tuples of space functions that satisfy them. For example, if we were to chose $\delta_{\emptyset} = \lambda_{\top}$, $\delta_{\{i\}} = \mathfrak{s}_i$ for every $i \in G$, and $\delta_I = \lambda_{\perp}$ whenever $|I| > 1$ then D1, D2 and D3 would be trivially met. But these space functions would not capture our intended meaning of distributed spaces: E.g., we would have *true* $\sqsupseteq \delta_I(e)$ for every e thus implying that any e could be distributed in the empty information *true* amongst the agents in $I \neq \emptyset$.

Nevertheless, the following theorem states that $(\Delta_I)_{I\subseteq G}$ could have been equivalently defined as the greatest space functions satisfying Properties D1–D3.

▶ **Theorem 20** (Max gdc). *Let $(\Delta_I)_{I\subseteq G}$ be the distributed spaces of $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$. Then $(\Delta_I)_{I\subseteq G}$ is a gdc of $\mathbf{C}$ and if $(\delta_I)_{I\subseteq G}$ is a gdc of $\mathbf{C}$ then $\delta_I \sqsubseteq_s \Delta_I$ for each $I \subseteq G$.*

Let us illustrate the use of Properties D1-D3 in Thm. 20 with the following example.

▶ **Example 21.** Let $c = \mathfrak{s}_1(a)\sqcup\mathfrak{s}_2(a \to b)\sqcup\mathfrak{s}_3(b \to e)$ as in Ex. 7. We want to prove $c \sqsupseteq \Delta_I(e)$ for $I = \{1, 2, 3\}$. From D.2 we have $c = \Delta_{\{1\}}(a) \sqcup \Delta_{\{2\}}(a \to b) \sqcup \Delta_{\{3\}}(b \to e)$. We can then use D.3 to obtain $c \sqsupseteq \Delta_I(a) \sqcup \Delta_I(a \to b) \sqcup \Delta_I(b \to e)$. Finally, by D.1 and Proposition 3 we infer $c \sqsupseteq \Delta_I(a \sqcup (a \to b) \sqcup (b \to e)) \sqsupseteq \Delta_I(e)$, thus $c \sqsupseteq \Delta_I(e)$ as wanted. Now consider our counter-example in Ex. 10 with $d = \mathfrak{s}_1(b) \sqcap \mathfrak{s}_2(b)$. We wish to prove $d \sqsupseteq \Delta_I(b)$ for $I = \{1, 2\}$. I.e., that b can be derived from d as being in a space of a member of {1,2}. Using D.1 and D.3 we obtain $d \sqsupseteq d' = \Delta_{\{1\}}(b) \sqcap \Delta_{\{2\}}(b) \sqsupseteq \Delta_{\{1,2\}}(b) \sqcap \Delta_{\{1,2\}}(b) = \Delta_{\{1,2\}}(b)$ as wanted.

The characterization of distributed spaces by Thm. 20 provide us with a convenient proof method: E.g. to prove that a tuple $F = (f_I)_{I\subseteq G}$ equals $(\Delta_I)_{I\subseteq G}$, it suffices to show that the tuple is a gdc and that $f_I \sqsupseteq_{\mathsf{s}} \Delta_I$ for all $I \subseteq G$. We use this mechanism in Section 5.

4.4 Group Projections

As promised in Section 4.1 we now give a definition of *Group Projection.* The function $\Pi_I(c)$ extracts exactly all information that the group I may have distributed w.r.t. c.

▶ **Definition 22** (Group Projection). *Let $(\Delta_I)_{I\subseteq G}$ be the distributed spaces of an scs $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$. Given the set $I \subseteq G$, the I-group projection of $c \in Con$ is defined as $\Pi_I(c) \stackrel{\text{def}}{=} \bigsqcup\{e \mid c \sqsupseteq \Delta_I(e)\}$. We say that e is I-group derivable from c iff $\Pi_I(c) \sqsupseteq e$.*

Much like space functions and agent projections, group projections and distributed spaces also form a pleasant correspondence: a Galois connection [3].

▶ **Proposition 23.** *Let $(\Delta_I)_{I\subseteq G}$ be the distributed spaces of $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$. For every $c, e \in Con$, (1) $c \sqsupseteq \Delta_I(e)$ iff $\Pi_I(c) \sqsupseteq e$, (2) $\Pi_I(c) \sqsupseteq \Pi_J(c)$ if $J \subseteq I$, and (3) $\Pi_I(c) \sqsupseteq \pi_I(c)$.*

The first property in Prop. 23, a Galois connection, states that we can conclude from c that e is in the distributed space of I exactly when e is I-group derivable from c. The second says that the bigger the group, the bigger the projection. The last property says that whatever is I-join derivable is I-group derivable, although the opposite is not true as shown in Ex. 10.

4.5 Group Compactness

Suppose that an *infinite* group of agents I can derive e from c (i.e., $c \sqsupseteq \Delta_I(e)$). A legitimate question is whether there exists a *finite* sub-group J of agents from I that can also derive e from c. The following theorem provides a positive answer to this question provided that e is a compact element (see Section 2) and I-join derivable from c.

▶ **Theorem 24** (Group Compactness). *Let $(\Delta_I)_{I\subseteq G}$ be the distributed spaces of an scs $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$. Suppose that $c \sqsupseteq \Delta_I(e)$. If e is compact and I-join derivable from c then there exists a finite set $J \subseteq I$ such that $c \sqsupseteq \Delta_J(e)$.*

We conclude this section with the following example of group compactness.

▶ **Example 25.** Consider the example with infinitely many agents in Ex. 7. We have $c' = \bigsqcup_{i\in\mathbb{N}} \mathfrak{s}_i(a_i)$ for some increasing chain $a_0 \sqsubseteq a_1 \sqsubseteq \ldots$ and e' s.t. $e' \sqsubseteq \bigsqcup_{i\in\mathbb{N}} a_i$. Notice that $c' \sqsupseteq \Delta_{\mathbb{N}}(e')$ and $\pi_{\mathbb{N}}(c') \sqsupseteq e'$. Hence e' is $\mathbb{N}$-join derivable from c'. If e' is compact, by Thm. 24 there must be a finite subset $J \subseteq \mathbb{N}$ such that $c' \sqsupseteq \Delta_J(e')$.

5 Computing Distributed Information

Let us consider a *finite* scs $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i\in G})$ with distributed spaces $(\Delta_I)_{I\subseteq G}$. By finite scs we mean that Con and G are finite sets. Let us consider the problem of computing Δ_I: Given a set $\{\mathfrak{s}_i\}_{i\in I}$ of space functions, we wish to find the greatest space function f such that $f \sqsubseteq \mathfrak{s}_i$ for all $i \in I$ (see Def. 14).

Because of the finiteness assumption, the above problem can be rephrased in simpler terms: *Given a finite lattice L and a finite set S of join-homomorphisms on L, find the greatest join-homomorphism below all the elements of S.* Even in small lattices with four elements and two space functions, finding such greatest function may not be immediate, e.g., for $S = \{\mathfrak{s}_1, \mathfrak{s}_2\}$ and the lattice in Fig. 1 the answer is given Fig. 2b.

In this section we shall use the theory developed in previous sections to help us find algorithms for this problem. Recall from Def. 14 and Lemma 12 that Δ_I equals the following

$$max\{f \in \mathcal{S}(\mathbf{C}) \mid f \sqsubseteq \mathfrak{s}_i \text{ for all } i \in I\} = \bigsqcup\{f \in \mathcal{S}(\mathbf{C}) \mid f \sqsubseteq \mathfrak{s}_i \text{ for all } i \in I\} = \bigsqcap\{\mathfrak{s}_i \mid i \in I\}$$

A *naive (meet-based) approach* would be to compute $\Delta_I(c)$ by taking the point-wise meet construction $\sigma_I(c) \stackrel{\text{def}}{=} \bigsqcap\{\mathfrak{s}_i(c) \mid i \in I\}$ for each $c \in Con$. But this does not work in general since $\Delta_I(c) = \bigsqcap\{\mathfrak{s}_i \mid i \in I\}(c)$ is not necessarily equal to $\sigma_I(c) = \bigsqcap\{\mathfrak{s}_i(c) \mid i \in I\}$. In fact $\sigma_I \sqsupseteq_{\mathsf{s}} \Delta_I$ but σ_I may not even be a space function as shown in Fig. 3a.

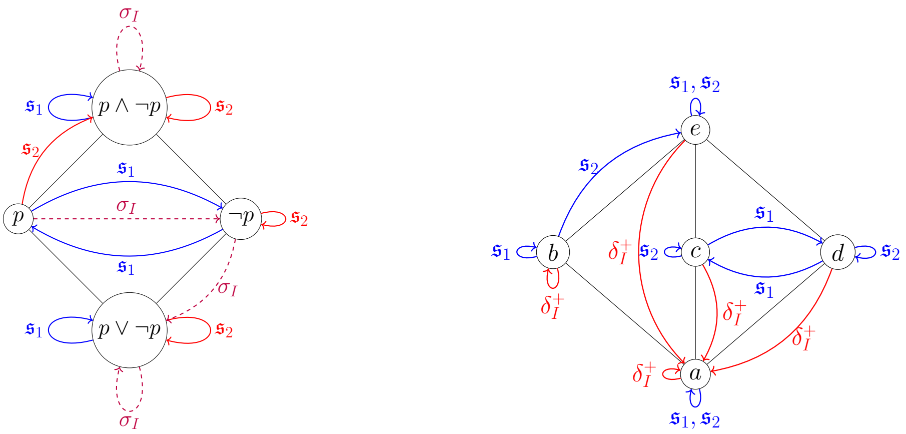
A *brute force (join-based)* solution to computing $\Delta_I(c)$ can be obtained by generating the set $\{f(c) \mid f \in \mathcal{S}(\mathbf{C})$ and $f \sqsubseteq \mathfrak{s}_i$ for all $i \in I\}$ and taking its join. This approach works since the join of a set of space functions S can be computed point-wise: $(\bigsqcup S)(c) = \bigsqcup\{f(c) | f \in S\}$. However, the number of such functions in $\mathcal{S}(\mathbf{C})$ can be at least factorial in the size of Con. For constraint frames, which under the finite assumption coincides with distributive lattices, the size of $\mathcal{S}(\mathbf{C})$ can be non-polynomial in the size of Con.

▶ **Proposition 26** (Lower Bounds on Number of Space Functions). *For every $n \geq 2$, there exists a cs $\mathbf{C} = (Con, \sqsubseteq)$ such that $|\mathcal{S}(\mathbf{C})| \geq (n-2)!$ and $n = |Con|$. For every $n \geq 1$, there exists a constraint frame $\mathbf{C} = (Con, \sqsubseteq)$ such that $|\mathcal{S}(\mathbf{C})| \geq n^{\log_2 n}$ and $n = |Con|$.*

Nevertheless, in the following sections we shall be able to exploit order theoretical results and properties of distributed spaces to compute $\Delta_I(c)$ for every $c \in Con$ in polynomial time in the size of Con. The first approach uses the inherent compositional nature of Δ_I in distributed lattices. The second approach uses the above-mentioned σ as a suitable upper bound to compute Δ_I by approximating it from above.

5.1 Distributed Spaces in Distributed Lattices

Here we shall illustrate some pleasant compositionality properties of distributed spaces that can be used for computing Δ_I in distributed lattices (constraint frames). These properties capture the intuition that just like *distributed information* of a group I is the collective



(a) For $I = \{1,2\}$, $\sigma_I(c) = \bigsqcap_{i \in I} \mathfrak{s}_i(c)$ is not a space function: $\sigma_I(p \sqcup \neg p) \neq \sigma_I(p) \sqcup \sigma_I(\neg p)$.

(b) For $I = \{1,2\}$, δ_I^+ (Lemma 27) is not a space function: $\delta_I^+(b) \sqcup \delta_I^+(e) = b \neq a = \delta_I^+(b \sqcup e)$.

Figure 3 Counter-examples over lattice $\mathbf{M}_2$ (a) and the non- distributive lattice $\mathbf{M}_3$ (b).

information from all its members, it is also the collective information of its subgroups. The following results can be used to produce algorithms to compute $\Delta_I(c)$.

We use X^J to denote the set of tuples $(x_j)_{j \in J}$ of elements $x_j \in X$ for each $j \in J$.

▶ **Lemma 27.** *Let $(\Delta_I)_{I \subseteq G}$ be the distributed spaces of a finite scs $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$. Suppose that $(Con, \sqsubseteq)$ is a constraint frame. Let $\delta_I^+ : Con \to Con$, with $I \subseteq G$, be the function $\delta_I^+(c) \stackrel{\text{def}}{=} \bigsqcap\{\bigsqcup_{i \in I} \mathfrak{s}_i(a_i) \mid (a_i)_{i \in I} \in Con^I \text{ and } \bigsqcup_{i \in I} a_i \sqsupseteq c\}$. Then $\Delta_I = \delta_I^+$.*

The above lemma basically says that $\Delta_I(c)$ is the greatest information below all possible combinations of information in the spaces of the agents in I that derive c. The proof that $\delta_I^+ \sqsupseteq_{\mathsf{s}} \Delta_I$ uses the fact that space functions preserve joins. The proof that $\delta_I^+ \sqsubseteq_{\mathsf{s}} \Delta_I$ proceeds by showing that $(\delta_I^+)_{I \subseteq G}$ is a group distribution candidate (Def. 19). Distributivity of the lattice $(Con, \sqsubseteq)$ is crucial for this direction. In fact without it $\Delta_I = \delta_I^+$ does not necessarily hold as shown by the following counter-example.

▶ **Example 28.** Consider the non-distributive lattice $\mathbf{M}_3$ and the space functions $\mathfrak{s}_1$ and $\mathfrak{s}_2$ in Fig. 3b. We obtain $\delta_I^+(b \sqcup c) = \delta_I^+(e) = a$ and $\delta_I^+(b) \sqcup \delta_I^+(c) = b \sqcup a = b$. Then, $\delta_I^+(b \sqcup c) \neq \delta_I^+(b) \sqcup \delta_I^+(c)$, i.e., δ_I^+ is not a space function.

Lemma 27 can be used to prove the following theorem which intuitively characterizes the information of a group from that of its subgroups. Each of the following results will be used to generate algorithms to compute $\Delta_I(c)$, each an improvement on the previous one.

▶ **Theorem 29.** *Let $(\Delta_I)_{I \subseteq G}$ be the distributed spaces of a finite scs $\mathbf{C} = (Con, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$. Suppose that $(Con, \sqsubseteq)$ is a constraint frame. Let $J, K \subseteq G$ be two groups such that $I = J \cup K$. Then the following equalities hold:*

1. $$\Delta_I(c) = \bigsqcap\{\Delta_J(a) \sqcup \Delta_K(b) \mid a, b \in Con \text{ and } a \sqcup b \sqsupseteq c\}. \tag{1}$$
2. $$\Delta_I(c) = \bigsqcap\{\Delta_J(a) \sqcup \Delta_K(a \to c) \mid a \in Con\}. \tag{2}$$
3. $$\Delta_I(c) = \bigsqcap\{\Delta_J(a) \sqcup \Delta_K(a \to c) \mid a \in Con \text{ and } a \sqsubseteq c\}. \tag{3}$$

The above properties bear witness to the inherent compositional nature of our notion of distributed space. This nature will be exploited by the algorithms below. The first property in Thm. 29 essentially reformulates Lemma 27 in terms of subgroups rather than agents. It can be proven by replacing $\Delta_J(a)$ and $\Delta_K(b)$ by $\delta_J^+(a)$ and $\delta_K^+(b)$, defined in Lemma 27 and using distributivity of joins over meets. The second and third properties in Thm. 29 are pleasant simplifications of the first using heyting implication. These properties realize the intuition that by joining the information a and $a \to c$ of their subgroups, the group I can obtain c.

5.2 Algorithms for Distributed Lattices

Recall that $\lambda_\top$ represents the empty distributed space (see Def. 13). Given a finite scs $\mathbf{C} = (\mathit{Con}, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$ with distributed spaces $(\Delta_I)_{I \subseteq G}$, the recursive function DeltaPart3(I, c) in Algorithm 1 computes $\Delta_I(c)$ for any given $c \in \mathit{Con}$. Its correctness, assuming that $(\mathit{Con}, \sqsubseteq)$ is a constraint frame (i.e., a distributed lattice), follows from Thm. 29(3). Termination follows from the finiteness of $\mathbf{C}$ and the fact the sets J and K in the recursive calls form a partition of I. Notice that we select a partition (in halves) rather than any two sets K, J satisfying the condition $I = J \cup K$ to avoid significant recalculation.

Algorithm 1 Function DeltaPart3(I, c) computes $\Delta_I(c)$.

```
1: function DeltaPart3($I, c$)
2:     if $I = \emptyset$ then
3:         return $\lambda_\top(c)$
4:     else if $I = \{i\}$ then
5:         return $\mathfrak{s}_i(c)$
6:     else
7:         $\{J, K\} \leftarrow$ Partition($I$)    ▷ returns a partition $\{J, K\}$ of $I$ s.t., $|J| = \lfloor |I|/2 \rfloor$
8:         return $\bigsqcap\{$DeltaPart3$(J, a) \sqcup$ DeltaPart3$(K, a \to c) \mid a \in \mathit{Con}$ and $a \sqsubseteq c\}$.
```

Algorithms. DeltaPart3(I, c) computes $\Delta_I(c)$ using Thm. 29(3). By modifying Line 8 with the corresponding meet operations, we obtain two variants of DeltaPart3 that use, instead of Thm. 29(3), the Properties Thm. 29(1) and Thm. 29(2). We call them DeltaPart1 and DeltaPart2. Finally, we also obtain a non-recursive algorithm that outputs $\Delta_I(c)$ by computing $\delta_I^+(c)$ in Lemma 27 in the obvious way: Computing the meet of elements of the form $\bigsqcup_{i \in I} \mathfrak{s}_i(a_i)$ for every tuple $(a_i)_{i \in I}$ such that $\bigsqcup_{i \in I} a_i \sqsupseteq c$. We call it Delta+.

Worst-case time complexity. We assume that binary distributive lattice operations $\sqcap$, $\sqcup$, and $\to$ are computed in $O(1)$ time. We also assume a fixed group I of size $m = |I|$ and express the time complexity for computing Δ_I in terms of $n = |\mathit{Con}\,|$, the size of the set of constraints. The above-mentioned algorithms compute the value $\Delta_I(c)$. The worst-case time complexity for computing the function Δ_I is in (1) $O(mn^{1+m})$ using Delta+, (2) $O(mn^{1+2\log_2 m})$ using DeltaPart1, and (3) $O(mn^{1+\log_2 m})$ using DeltaPart2 and DeltaPart3.

5.3 Algorithm for Arbitrary Lattices

Let $(\Delta_I)_{I \subseteq G}$ be the distributed spaces of a finite scs $\mathbf{C} = (\mathit{Con}, \sqsubseteq, (\mathfrak{s}_i)_{i \in G})$. The maximum space function Δ_I under a collection $\{\mathfrak{s}_i\}_{i \in I}$ can be computed by successive approximations, starting with some (not necessarily space) function known to be less than all $\{\mathfrak{s}_i\}_{i \in I}$. Assume a self map $\sigma : Con \to Con$ such that $\sigma \sqsupseteq \Delta_I$ and, for all $i \in I$, $\sigma \sqsubseteq \mathfrak{s}_i$. A good starting point is $\sigma(u) = \bigsqcap\{\mathfrak{s}_i(u) \mid i \in I\}$, for all $u \in Con$. By definition of $\sqcap$, $\sigma(u)$ is the biggest function under all functions in $\{\mathfrak{s}_i\}_{i \in I}$, hence $\sigma \sqsupseteq \Delta_I$. The algorithm computes decreasing upper bounds of Δ_I by correcting σ values not conforming to the space function property $\sigma(u) \sqcup \sigma(v) = \sigma(u \sqcup v)$. The correction decreases σ and maintains the invariant $\sigma \sqsupseteq \Delta_I$.

There are two ways of correcting σ values: (1) when $\sigma(u) \sqcup \sigma(v) \sqsubset \sigma(u \sqcup v)$, assign $\sigma(u \sqcup v) \leftarrow \sigma(u) \sqcup \sigma(v)$ and (2) when $\sigma(u) \sqcup \sigma(v) \not\sqsubseteq \sigma(u \sqcup v)$, assign $\sigma(u) \leftarrow \sigma(u) \sqcap \sigma(u \sqcup v)$ and also $\sigma(v) \leftarrow \sigma(v) \sqcap \sigma(u \sqcup v)$. It can be shown that the assignments in both cases should decrease σ while preserving the $\sigma \sqsupseteq \Delta_I$ invariant.

The procedure (see Algorithm 2) loops through pairs $u, v \in Con$ while there is some pair satisfying cases (1) or (2) above for the current σ. When there is, it updates σ as mentioned before. At the end of the loop all $u, v \in Con$ pairs satisfy the space function property. By the invariant mentioned above, this means $\sigma = \Delta_I$.

Algorithm 2 DeltaGen finds Δ_I.

```
$\sigma(u) \leftarrow \bigsqcap\{\mathfrak{s}_i(u) \mid i \in I\}$                              ▷ for all $u \in Con$
while $u, v \in Con \wedge \sigma(u) \sqcup \sigma(v) \neq \sigma(u \sqcup v)$ do
    if $\sigma(u) \sqcup \sigma(v) \sqsubset \sigma(u \sqcup v)$ then                          ▷ case (1)
        $\sigma(u \sqcup v) \leftarrow \sigma(u) \sqcup \sigma(v)$
    else                                                                                        ▷ case (2)
        $\sigma(u) \leftarrow \sigma(u) \sqcap \sigma(u \sqcup v)$
        $\sigma(v) \leftarrow \sigma(v) \sqcap \sigma(u \sqcup v)$
```

Assume a fixed group I of size $m = |I|$ and that $\sqcap$ and $\sqcup$ are computed in $O(1)$ time. The complexity of the initialization of DeltaGen is $O(nm)$ with $n = |Con|$. Each element in Con can be decreased at most n times. Identifying an element to be decreased (in the test of the loop) takes $O(n^2)$. Since there are n^2 possible decreases, worst time complexity of the loop is in $O(n^4)$.

6 Conclusions and Related Work

We developed semantic foundations and provided algorithms for reasoning about the distributed information of groups in multi-agents systems. We plan to develop similar techniques for reasoning about other group phenomena in multi-agent systems from social sciences and computer music such as group polarization [4] and group improvisation [17].

The closest related work is that of [15] (and its extended version [16]) which introduces spatial constraint systems (scs) for the semantics of a spatial ccp language. Their work is confined to a finite number of agents and to reasoning about agents individually rather than as groups. We added the continuity requirement to the space functions of [15] to be able to reason about possibly infinite groups. In [7, 8, 9, 10] scs are used to reason about beliefs, lies and other epistemic utterances but also restricted to a finite number of agents and individual, rather than group, behaviour of agents.

Our work is inspired by the epistemic concept of distributed knowledge [5]. Knowledge in distributed systems was discussed in [11], based on interpreting distributed systems using Hintikka's notion of possible worlds. In this definition of distributed knowledge, the system designer ascribes knowledge to processors (agents) in each global state (a processor's local state). In [12] the authors present a general framework to formalize the knowledge of a group of agents, in particular the notion of distributed knowledge. The authors consider distributed knowledge as knowledge that is distributed among the agents belonging to a given group, without any individual agent necessarily having this knowledge. In [13] the authors study knowledge and common knowledge in situations with infinitely many agents. The authors highlight the importance of reasoning about infinitely many agents in situations where the number of agents is not known in advance. Their work does not address distributed knowledge but points out potential technical difficulties in their future work.

References

1 Samson Abramsky and Achim Jung. Domain Theory. In Samson Abramsky et al., editors, *Handbook of Logic in Computer Science*, volume 3, pages 1–168. Oxford University Press, 1994.

2 Frank S. Boer, Alessandra Di Pierro, and Catuscia Palamidessi. Nondeterminism and infinite computations in constraint programming. *Theoretical Computer Science*, pages 37–78, 1995.

3 Brian A Davey and Hilary A Priestley. *Introduction to lattices and order*. Cambridge university press, 2nd edition, 2002.

4 Joan-María Esteban and Debraj Ray. On the Measurement of Polarization. *Econometrica*, 62(4):819–851, 1994.

5 Ronald Fagin, Joseph Y Halpern, Yoram Moses, and Moshe Y Vardi. *Reasoning about knowledge*. MIT press Cambridge, 4th edition, 1995.

6 Gerhard Gierz, Karl Heinrich Hofmann, Klaus Keimel, Jimmie D. Lawson, Michael Mislove, and Dana S. Scott. *Continuous lattices and domains*. Cambridge University Press, 2003.

7 Michell Guzmán, Stefan Haar, Salim Perchy, Camilo Rueda, and Frank Valencia. Belief, Knowledge, Lies and Other Utterances in an Algebra for Space and Extrusion. *Journal of Logical and Algebraic Methods in Programming*, 2016. URL: `https://hal.inria.fr/hal-01257113`.

8 Michell Guzman, Salim Perchy, Camilo Rueda, and Frank Valencia. Deriving Inverse Operators for Modal Logic. In *Theoretical Aspects of Computing – ICTAC 2016*, volume 9965 of *Lecture Notes in Computer Science*, pages 214–232. Springer, 2016. URL: `https://hal.inria.fr/hal-01328188`.

9 Michell Guzmán, Salim Perchy, Camilo Rueda, and Frank Valencia. Characterizing Right Inverses for Spatial Constraint Systems with Applications to Modal Logic. *Theoretical Computer Science*, 744(56–77), October 2018. URL: `https://hal.inria.fr/hal-01675010`.

10 Stefan Haar, Salim Perchy, Camilo Rueda, and Frank Valencia. An Algebraic View of Space/Belief and Extrusion/Utterance for Concurrency/Epistemic Logic. In *17th International Symposium on Principles and Practice of Declarative Programming (PPDP 2015)*, pages 161–172. ACM SIGPLAN, 2015. URL: `https://hal.inria.fr/hal-01256984`.

11 Joseph Y Halpern. Using reasoning about knowledge to analyze distributed systems. *Annual review of computer science*, 2(1):37–68, 1987.

12 Joseph Y Halpern and Yoram Moses. Knowledge and common knowledge in a distributed environment. *Journal of the ACM (JACM)*, 37(3):549–587, 1990.

13 Joseph Y Halpern and Richard A Shore. Reasoning about common knowledge with infinitely many agents. *Information and Computation*, 191(1):1–40, 2004.

14 Werner Hildenbrand. On economies with many agents. *Journal of Economic Theory*, 2(2):161–188, 1970.

15 Sophia Knight, Catuscia Palamidessi, Prakash Panangaden, and Frank D. Valencia. Spatial and Epistemic Modalities in Constraint-Based Process Calculi. In *CONCUR 2012 - 23rd International Conference on Concurrency Theory*, volume 7454 of *Lecture Notes in Computer Science*, pages 317–332. Springer, 2012. URL: `https://hal.archives-ouvertes.fr/hal-00761116`.

16 Sophia Knight, Prakash Panangaden, and Frank Valencia. Computing with Epistemic and Spatial Modalities. Submitted for journal publication, 2019.

17 Camilo Rueda and Frank Valencia. On validity in modelization of musical problems by CCP. *Soft Computing*, 8(9):641–648, 2004. `doi:10.1007/s00500-004-0390-7`.

18 Vijay A. Saraswat, Martin Rinard, and Prakash Panangaden. The Semantic Foundations of Concurrent Constraint Programming. In *Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '91, pages 333–352. ACM, 1991. `doi:10.1145/99583.99627`.

Robustness Against Transactional Causal Consistency

Sidi Mohamed Beillahi
Université de Paris, IRIF, CNRS, F-75013 Paris, France
beillahi@irif.fr

Ahmed Bouajjani
Université de Paris, IRIF, CNRS, F-75013 Paris, France
abou@irif.fr

Constantin Enea
Université de Paris, IRIF, CNRS, F-75013 Paris, France
cenea@irif.fr

Abstract

Distributed storage systems and databases are widely used by various types of applications. Transactional access to these storage systems is an important abstraction allowing application programmers to consider blocks of actions (i.e., transactions) as executing atomically. For performance reasons, the consistency models implemented by modern databases are weaker than the standard serializability model, which corresponds to the atomicity abstraction of transactions executing over a sequentially consistent memory. Causal consistency for instance is one such model that is widely used in practice.

In this paper, we investigate application-specific relationships between several variations of causal consistency and we address the issue of verifying automatically if a given transactional program is robust against causal consistency, i.e., all its behaviors when executed over an arbitrary causally consistent database are serializable. We show that programs without write-write races have the same set of behaviors under all these variations, and we show that checking robustness is polynomial time reducible to a state reachability problem in transactional programs over a sequentially consistent shared memory. A surprising corollary of the latter result is that causal consistency variations which admit incomparable sets of behaviors admit comparable sets of robust programs. This reduction also opens the door to leveraging existing methods and tools for the verification of concurrent programs (assuming sequential consistency) for reasoning about programs running over causally consistent databases. Furthermore, it allows to establish that the problem of checking robustness is decidable when the programs executed at different sites are finite-state.

2012 ACM Subject Classification Theory of computation → Program verification

Keywords and phrases Distributed Databases, Causal Consistency, Model Checking

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.30

Related Version Complete proofs can be found at `https://arxiv.org/abs/1906.12095`.

Funding This work is supported in part by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 678177).

1 Introduction

Distribution and replication are widely adopted in order to implement storage systems and databases offering performant and available services. The implementations of these systems must ensure consistency guarantees allowing to reason about their behaviors in an abstract and simple way. Ideally, programmers of applications using such systems would like to have strong consistency guarantees, i.e., all updates occurring anywhere in the system are seen immediately and executed in the same order by all sites. Moreover, application programmers also need an abstract mechanism such as transactions, ensuring that blocks

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 30; pp. 30:1–30:18
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

```
t1 [z = 1          t3 [x = 2
      x = 1]   ||       r1 = z] //0
t2 [y = 1]         t4 [r2 = y  //1
                        r3 = x] //2
```

(a) CCv but not CM.

```
t1 [x = 1]             t3 [x = 2]
t2 [r1 = x] //2   ||   t4 [r2 = x] //1
```

(b) CM but not CCv.

```
                     t2 [x = 1]
t1 [x = 2]     ||    t3 [r1 = x] //2
                     t4 [r2 = x] //1
```

(c) CC but not CM nor CCv.

Figure 1 Computations showing the relation between CC, CCv and CM. Transactions are delimited using brackets, same site transactions are aligned vertically, and read values are given in comments.

of actions (writes and reads) of a site can be considered as executing atomically without interferences from actions of other sites. For transactional programs, the consistency model offering strong consistency is *serializability* [37], i.e., every computation of a program is equivalent to another one where transactions are executed serially one after another without interference. In the non-transactional case this model corresponds to *sequential consistency* (SC) [31]. However, while serializability and SC are easier to apprehend by application programmers, their enforcement (by storage systems implementors) requires the use of global synchronization between all sites, which is hard to achieve while ensuring availability and acceptable performances [24, 25]. For this reason, modern storage systems ensure weaker consistency guarantees. In this paper, we are interested in studying *causal consistency* [30].

Causal consistency is a fundamental consistency model implemented in several production databases, e.g., AntidoteDB, CockroachDB, and MongoDB, and extensively studied in the literature [7, 23, 33, 34, 39]. Basically, when defined at the level of actions, it guarantees that every two causally related actions, say a_1 is causally before (i.e., it has an influence on) a_2, are executed in that same order, i.e., a_1 before a_2, by all sites. The sets of updates visible to different sites may differ and read actions may return values that cannot be obtained in SC executions. The definition of causal consistency can be lifted to the level of transactions, assuming that transactions are visible to a site in their entirety (i.e., all their updates are visible at the same time), and they are executed by a site in isolation without interference from other transactions. In comparison to serializability, causal consistency allows that conflicting transactions, i.e., which read or write to a common location, be executed in different orders by different sites as long as they are not causally related. Actually, we consider three variations of causal consistency introduced in the literature, weak causal consistency (CC) [38, 14], causal memory (CM) [3, 38], and causal convergence (CCv) [18].

The weakest variation of causal consistency, namely CC, allows speculative executions and roll-backs of transactions which are not causally related (concurrent). For instance, the computation in Fig. 1c is only feasible under CC: the site on the right applies t2 after t1 before executing t3 and roll-backs t2 before executing t4. CCv and CM offer more guarantees. CCv enforces a total *arbitration* order between all transactions which defines the order in which delivered concurrent transactions are executed by *every* site. This guarantees that all sites reach the same state when all transactions are delivered. CM ensures that *all* values read by a site can be explained by an interleaving of transactions consistent with the causal order, enforcing thus PRAM consistency [32] on top of CC. Contrary to CCv, CM allows that two sites diverge on the ordering of concurrent transactions, but both models do not allow roll-backs of concurrent transactions. Thus, CCv and CM are incomparable in terms of computations they admit. The computation in Fig. 1a is not admitted by CM because there is no interleaving of those transactions that explains the values read by the site on the right: reading 0 from z implies that the transactions on the left must be applied after t3 while reading 1 from y implies that both t1 and t2 are applied before t4 which contradicts reading 2 from x. However, this computation is possible under CCv because t1 can be delivered to the right after executing t3 but arbitrated before t3, which implies that the write to x in t1 will be lost. The CM computation in Fig. 1b is not possible under CCv because there is no arbitration order that could explain both reads from x.

As a first contribution of our paper, we show that the three causal consistency semantics coincide for transactional programs containing no *write-write races*, i.e., concurrent transactions writing on a common variable. We also show that if a transactional program has a write-write race under one of these semantics, then it must have a write-write race under any of the other two semantics. This property is rather counter-intuitive since CC is strictly weaker than both CCv and CM, and CCv and CM are incomparable (in terms of admitted behaviors). Notice that each of the computations in Figures 1b, 1a, and 1c contains a write-write race which explains why none of these computations is possible under all three semantics.

Then, we investigate the problem of checking *robustness* of application programs against causal consistency relaxations: Given a program P and a causal consistency variation X, we say that P is robust against X if the set of computations of P when running under X is the same as its set of computations when running under *serializability*. This means that it is possible to reason about the behaviors of P assuming the simpler serializability model and no additional synchronization is required when P runs under X such that it maintains all the properties satisfied under serializability. Checking robustness is not trivial, it can be seen as a form of checking program equivalence. However, the equivalence to check is between two versions of the same program, obtained using two different semantics, one more permissive than the other one. The goal is to check that this permissiveness has actually no effect on the particular program under consideration. The difficulty in checking robustness is to apprehend the extra behaviors due to the reorderings introduced by the relaxed consistency model w.r.t. serializability. This requires a priori reasoning about complex order constraints between operations in arbitrarily long computations, which may need maintaining unbounded ordered structures, and make the problem of checking robustness hard or even undecidable.

We show that verifying robustness of transactional programs against causal consistency can be reduced in polynomial time to the reachability problem in concurrent programs over SC. This allows to reason about distributed applications running on causally consistent storage systems using the existing verification technology and it implies that the robustness problem is decidable for finite-state programs; the problem is PSPACE-complete when the number of sites is fixed, and EXPSPACE-complete otherwise. This is the first result on the decidability and complexity of verifying robustness against causal consistency. In fact, the problem of verifying robustness has been considered in the literature for several consistency models of distributed systems, including causal consistency [12, 16, 17, 20, 35]. These works provide (over- or under-)approximate analyses for checking robustness, but none of them provides precise (sound and complete) algorithmic verification methods for solving this problem, nor addresses its decidability and complexity.

The approach we adopt for tackling this verification problem is based on a precise characterization of the set of robustness violations, i.e., executions that are causally consistent but not serializable. For both CCv and CM, we show that it is sufficient to search for a special type of robustness violations, that can be simulated by serial (SC) computations of an instrumentation of the original program. These computations maintain the information needed to recognize the pattern of a violation that would have occurred in the original program under a causally consistent semantics (executing the same set of operations). A surprising consequence of these results is that a program is robust against CM iff it is robust against CC, and robustness against CM implies robustness against CCv. This shows that the causal consistency variations we investigate can be incomparable in terms of the admitted behaviors, but comparable in terms of the robust applications they support.

2 Causal Consistency

Program syntax. We consider a simple programming language where a program is parallel composition of *processes* distinguished using a set of identifiers $\mathbb{P}$. Each process is a sequence of *transactions* and each transaction is a sequence of *labeled instructions*. Each transaction starts with a `begin` instruction and finishes with an `end` instruction. Each other instruction is either an assignment to a process-local *register* from a set $\mathbb{R}$ or to a *shared variable* from a set $\mathbb{V}$, or an `assume` statement. The assignments use values from a data domain $\mathbb{D}$. An assignment to a register $\langle reg \rangle := \langle var \rangle$ is called a *read* of $\langle var \rangle$ and an assignment to a shared variable $\langle var \rangle := \langle reg\text{-}expr \rangle$ is called a *write* to $\langle var \rangle$ ($\langle reg\text{-}expr \rangle$ is an expression over registers). The statement `assume` $\langle bexpr \rangle$ blocks the process if the Boolean expression $\langle bexpr \rangle$ over registers is false. Each instruction is followed by a `goto` statement which defines the evolution of the program counter. Multiple instructions can be associated with the same label which allows us to write non-deterministic programs and multiple `goto` statements can direct the control to the same label which allows us to mimic imperative constructs like loops and conditionals. We assume that the control cannot pass from one transaction to another without going as expected through `begin` and `end` instructions.

Causal memory (CM) semantics. Informally, the semantics of a program under causal memory is defined as follows. The shared variables are replicated across each process, each process maintaining its own local valuation of these variables. During the execution of a transaction in a process, the shared-variable writes are stored in a *transaction log* which is visible only to the process executing the transaction and which is broadcasted to all the other processes at the end of the transaction[1]. To read a shared variable x, a process p first accesses its transaction log and takes the last written value on x, if any, and then its own valuation of the shared variables, if x was not written during the current transaction. Transaction logs are delivered to every process in an order consistent with the *causal delivery* relation between transactions, i.e., the transitive closure of the union of the *program order* (the order in which transactions are executed by a process), and the *delivered-before* relation (a transaction t_1 is delivered-before a transaction t_2 iff the log of t_2 has been delivered at the process executing t_1 before t_1 starts). By an abuse of terminology, we call this property *causal delivery*. Once a transaction log is delivered, it is immediately applied on the shared-variable valuation of the receiving process. Also, no transaction log can be delivered to a process p while p is executing another transaction, we call this property *transaction isolation*.

Causal convergence (CCv) semantics. Compared to causal memory, causal convergence ensures eventual consistency of process-local copies of the shared variables. Each transaction log is associated with a timestamp and a process applies a write on some variable x from a transaction log only if it has a timestamp larger than the timestamps of all the transaction logs it has already applied and that wrote the same variable x. For simplicity, we assume that the transaction identifiers play the role of timestamps, which are totally ordered according to some relation $<$. `CCv` satisfies both *causal delivery* and *transaction isolation* as well.

Weak causal consistency (CC) semantics. Compared to the previous semantics, `CC` allows that reads of the same process observe concurrent writes as executing in different orders. Each process maintains a *set* of values for each shared variable, and a read returns any one

[1] For simplicity, we assume that every transaction commits. The effects of aborted transactions shouldn't be visible to any process.

$$\begin{array}{l}\mathsf{begin}(p1,t1)\\ \quad\mathsf{isu}(p1,t1,x,1)\\ \mathsf{end}(p1,t1)\end{array} \cdot \begin{array}{l}\mathsf{begin}(p2,t3)\\ \quad\mathsf{isu}(p2,t3,x,2)\\ \mathsf{end}(p2,t3)\end{array} \cdot \mathsf{del}(p1,t3) \cdot \mathsf{del}(p2,t1) \cdot \begin{array}{l}\mathsf{begin}(p2,t4)\\ \quad\mathsf{ld}(p2,t4,x,1)\\ \mathsf{end}(p2,t4)\end{array} \cdot \begin{array}{l}\mathsf{begin}(p1,t2)\\ \quad\mathsf{ld}(p1,t2,x,2)\\ \mathsf{end}(p1,t2)\end{array}$$

(a) CM execution of the program in Fig. 1b.

$$\begin{array}{l}\mathsf{begin}(p1,t1)\\ \quad\mathsf{isu}(p1,t1,z,1)\\ \quad\mathsf{isu}(p1,t1,x,1)\\ \mathsf{end}(p1,t1)\end{array} \cdot \begin{array}{l}\mathsf{begin}(p2,t3)\\ \quad\mathsf{isu}(p2,t3,x,2)\\ \quad\mathsf{ld}(p2,t3,z,0)\\ \mathsf{end}(p2,t3)\end{array} \cdot \mathsf{del}(p1,t3) \cdot \mathsf{del}(p2,t1) \cdot \begin{array}{l}\mathsf{begin}(p1,t2)\\ \quad\mathsf{isu}(p1,t2,y,1)\\ \mathsf{end}(p1,t2)\end{array} \cdot \mathsf{del}(p2,t2) \cdot \begin{array}{l}\mathsf{begin}(p2,t4)\\ \quad\mathsf{ld}(p2,t4,y,1)\\ \quad\mathsf{ld}(p2,t4,x,2)\\ \mathsf{end}(p2,t4)\end{array}$$

(b) CCv execution of the program in Figure 1a.

$$\begin{array}{l}\mathsf{begin}(p1,t1)\\ \quad\mathsf{isu}(p1,t1,x,2)\\ \mathsf{end}(p1,t1)\end{array} \cdot \begin{array}{l}\mathsf{begin}(p2,t2)\\ \quad\mathsf{isu}(p2,t2,x,1)\\ \mathsf{end}(p2,t2)\end{array} \cdot \mathsf{del}(p2,t1) \cdot \begin{array}{l}\mathsf{begin}(p2,t3)\\ \quad\mathsf{ld}(p2,t3,x,2)\\ \mathsf{end}(p2,t3)\end{array} \cdot \begin{array}{l}\mathsf{begin}(p2,t4)\\ \quad\mathsf{ld}(p2,t4,x,1)\\ \mathsf{end}(p2,t4)\end{array} \cdot \mathsf{del}(p1,t2)$$

(c) CC execution of the program in Figure 1c.

Figure 2 For readability, the sub-sequences of events delimited by begin and end are aligned vertically, the execution-flow advancing from left to right and top to bottom.

of these values non-deterministically. Transaction logs are associated with *vector clocks* [30] which represent the causal delivery relation, i.e., a transaction t_1 is before t_2 in causal-delivery iff the vector clock of t_1 is smaller than the vector clock of t_2. We assume that transactions identifiers play the role of vector clocks, which are partially ordered according to some relation $<$. When applying a transaction log on the shared-variable valuation of the receiving process, we only keep the values that were written by *concurrent* transactions (not related by causal delivery). CC satisfies both *causal delivery* and *transaction isolation*.

Program execution. The semantics of a program $\mathcal{P}$ under a causal consistency semantics $\mathsf{X} \in \{\mathtt{CCv},\ \mathtt{CM},\ \mathtt{CC}\}$ is defined using a labeled transition system $[\mathcal{P}]_{\mathsf{X}}$ where the set $\mathbb{E}\mathrm{v}$ of transition labels, called *events*, is defined by:

$$\mathbb{E}\mathrm{v} = \{\mathsf{begin}(p,t), \mathsf{ld}(p,t,x,v), \mathsf{isu}(p,t,x,v), \mathsf{del}(p,t), \mathsf{end}(p,t) : p \in \mathbb{P}, t \in \mathbb{T}, x \in \mathbb{V}, v \in \mathbb{D}\}$$

where begin and end label transitions corresponding to the start, resp., the end of a transaction, isu and ld label transitions corresponding to writing, resp., reading, a shared variable during some transaction, and del labels transitions corresponding to applying a transition log received from another process on the local copy of the shared variables. An event isu is called an *issue* while an event del is called a *store*. An execution of $[\mathcal{P}]_{\mathsf{X}}$ is a sequence of events $\rho = ev_1 \cdot ev_2 \cdot \ldots$ labeling the transitions. Fig. 2a shows an execution under CM. This satisfies transaction isolation since no transaction is delivered while another transaction is executing. The execution in Fig. 2a is not possible under CCv since $t4$ and $t2$ read 2 and 1 from x, respectively. This is possible only if $t1$ and $t3$ write x at $p2$ and $p1$, respectively, which contradicts the definition of CCv where we cannot have both $t1 < t3$ and $t3 < t1$. Fig. 2b shows an execution under CCv (we assume $t_1 < t_2 < t_3 < t_4$). Notice that $\mathsf{del}(p2,t1)$ did not result in an update of x because the timestamp t_1 is smaller than the timestamp of the last transaction that wrote x at p_2, namely t_3, a behavior that is not possible under CM. The two processes converge and store the same shared variable copy at the end of the execution. Fig. 2c shows an execution under CC, which is not possible under CCv and CM because $t3$ and $t4$ read 2 and 1, respectively. Since the transactions t_1 and t_2 are concurrent, p_2 stores both values 2 and 1 written by these transactions. A read of x can return any of these two values.

Execution summary. Let ρ be an execution under $\mathsf{X} \in \{\mathtt{CCv},\ \mathtt{CM},\ \mathtt{CC}\}$, a sequence τ of events $\mathsf{isu}(p,t)$ and $\mathsf{del}(p,t)$ with $p \in \mathbb{P}$ and $t \in \mathbb{T}$ is called a *summary of* ρ if it is obtained from ρ by substituting every sub-sequence of transitions in ρ delimited by a begin and an

end transition, with a single "macro-event" $\mathsf{isu}(p, t)$. For example, $\mathsf{isu}(p1, t1) \cdot \mathsf{isu}(p2, t3) \cdot \mathsf{del}(p1, t3) \cdot \mathsf{del}(p2, t1) \cdot \mathsf{isu}(p2, t4) \cdot \mathsf{isu}(p1, t2)$ is a summary of the execution in Fig. 2a.

We say that a transaction t in ρ performs an *external read* of a variable x if ρ contains an event $\mathsf{ld}(p, t, x, v)$ which is not preceded by a write on x of t (i.e., $\mathsf{isu}(p, t, x, v)$). Under CM and CC, a transaction t *writes* a variable x if ρ contains $\mathsf{isu}(p, t, x, v)$, for some v. In Fig. 2a, $t2$ and $t4$ perform external reads, and $t2$ writes to y. A transaction t executed by a process p *writes x at process $p' \neq p$* if t writes x and ρ contains $\mathsf{del}(p', t)$ (e.g., in Fig. 2a, $t1$ writes x at $p2$). Under CCv, we say that a transaction t executed by a process p *writes x at process $p' \neq p$* if t writes x and ρ contains an event $\mathsf{del}(p', t)$ which is not preceded by an event $\mathsf{del}(p', t')$ (or $\mathsf{isu}(p', t')$) with $t < t'$ and t' writing x (if it would be preceded by such an event then the write to x of t will be discarded). For example, in Fig. 2b, $t1$ does *not* write x at $p2$.

3 Write-Write Race Freedom

We say that an execution ρ has a *write-write race* on a shared variable x if there exist two concurrent transactions t_1 and t_2 that were issued in ρ and each transaction contains a write to the variable x. We call ρ write-write race free if there is no variable x such that ρ has a write-write race on x. Also, we say a program $\mathcal{P}$ is *write-write race free* under a consistency semantics $\mathsf{X} \in \{\mathtt{CCv}, \mathtt{CM}, \mathtt{CC}\}$ iff for every $\rho \in \mathbb{E}\mathrm{x}_{\mathsf{X}}(\mathcal{P})$, ρ is write-write race free.

We show that if a given program has a write-write race under one of the three causal consistency semantics then it must have a write-write race under the remaining two. The intuition behind this is that the three semantics coincide for programs without write-write races. Indeed, without concurrent transactions that write to the same variable, every process local valuation of a shared variable will be a singleton set under CC and no process will ever discard a write when applying an incoming transaction log under CCv.

▶ **Theorem 1.** *Given a program $\mathcal{P}$ and two consistency semantics $X, Y \in \{\mathtt{CCv}, \mathtt{CM}, \mathtt{CC}\}$, $\mathcal{P}$ has a write-write race under X iff $\mathcal{P}$ has a write-write race under Y.*

The following result shows that indeed, the three causal consistency semantics coincide for programs which are write-write race free under any one of these three semantics.

▶ **Theorem 2.** *Let $\mathcal{P}$ be a program. Then, $\mathbb{E}\mathrm{x}_{\mathtt{CC}}(\mathcal{P}) = \mathbb{E}\mathrm{x}_{\mathtt{CCv}}(\mathcal{P}) = \mathbb{E}\mathrm{x}_{\mathtt{CM}}(\mathcal{P})$ iff $\mathcal{P}$ has no write-write race under* CM*,* CCv*, or* CC*.*

4 Programs Robustness

4.1 Program Traces

We define an abstraction of executions satisfying transaction isolation, called *trace*. Intuitively, a trace forgets the order in which shared-variables are accessed inside a transaction and the order between transactions accessing different variables. The trace of an execution ρ is obtained by adding several standard relations between events in its summary which record the data-flow, e.g. which transaction wrote the value read by another transaction.

The *trace* of an execution ρ is a tuple $\mathsf{tr}(\rho) = (\tau, \mathsf{PO}, \mathsf{WR}, \mathsf{WW}, \mathsf{RW}, \mathsf{STO})$ where τ is the summary of ρ, PO is the *program order*, which relates any two issue events $\mathsf{isu}(p, t)$ and $\mathsf{isu}(p, t')$ that occur in this order in τ, WR is the *write-read* relation (also called *read-from*), which relates events of two transactions t and t' such that t writes a value that t' reads, WW is the *write-write* order (also called store-order), which relates events of two transactions that write to the same variable, and RW is the *read-write* relation (also called *conflict*), which relates events of two transactions t and t' such that t reads a value overwritten by t', and STO is the *same-transaction* relation, which relates events of the same transaction.

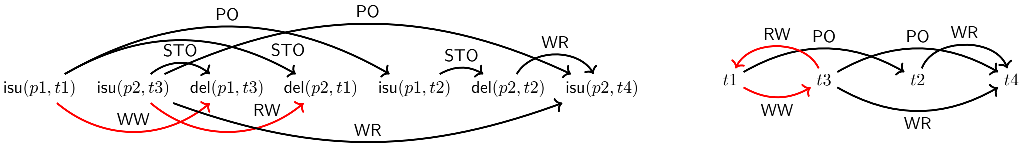


Figure 3 The trace of the execution in Fig. 2b and its transactional happens-before.

Formally, WR relates any two events $ev_1 \in \{\mathsf{isu}(p,t), \mathsf{del}(p,t)\}$ and $ev_2 = \mathsf{isu}(p,t')$ that occur in this order in τ such that t' performs an external read of x, and ev_1 is the last event in τ before ev_2 such that t writes x if $ev_1 = \mathsf{isu}(p,t)$, and t writes x at p if $ev_1 = \mathsf{del}(p,t)$ (we may use $\mathsf{WR}(x)$ to emphasize the variable x). Also, WW relates any two events $ev_1 \in \{\mathsf{isu}(p,t_1), \mathsf{del}(p,t_1)\}$ and $ev_2 \in \{\mathsf{isu}(p,t_2), \mathsf{del}(p,t_2)\}$ that occur in this order in τ provided that t_1 and t_2 both write the same variable x, and if ρ is an execution under causal convergence, then (1) if $ev_i = \mathsf{del}(p,t_i)$, for some $i \in \{1,2\}$, then t_i writes x at p, and (2) if $ev_1 \in \{\mathsf{isu}(p,t_1), \mathsf{del}(p,t_1)\}$ and $ev_2 = \mathsf{del}(p,t_2)$, then $t_1 < t_2$ (we may use $\mathsf{WW}(x)$ to emphasize the variable x). We also define $\mathsf{RW}(x) = \mathsf{WR}^{-1}(x); \mathsf{WW}(x)$ (we use ; to denote the standard composition of relations) and $\mathsf{RW} = \bigcup_{x\in\mathbb{V}} \mathsf{RW}(x)$. If a transaction t reads the initial value of x then $\mathsf{RW}(x)$ relates $\mathsf{isu}(p,t)$ to $\mathsf{isu}(p',t')$ of any other transaction t' which writes to x (i.e., $(\mathsf{isu}(p,t), \mathsf{isu}(p',t')) \in \mathsf{RW}(x)$). Finally, STO relates any event $\mathsf{isu}(p,t)$ with the set of events $\mathsf{del}(p',t)$, where $p \neq p'$.

Then, we define the *happens-before* relation HB as the transitive closure of the union of all the relations in the trace, i.e., $\mathsf{HB} = (\mathsf{PO} \cup \mathsf{WR} \cup \mathsf{WW} \cup \mathsf{RW} \cup \mathsf{STO})^+$. Since we reason about only one trace at a time, we may say that a trace is simply a summary τ, keeping the relations implicit. The trace of the CCv execution in Fig. 2b is shown on the left of Fig. 3. $\mathbb{T}\mathsf{r}(\mathcal{P})_{\mathsf{X}}$ denotes the set of traces of executions of a program $\mathcal{P}$ under $\mathsf{X} \in \{\mathtt{CCv}, \mathtt{CM}, \mathtt{CC}\}$.

The *causal order* CO of a trace $tr = (\tau, \mathsf{PO}, \mathsf{WR}, \mathsf{WW}, \mathsf{RW}, \mathsf{STO})$ is the transitive closure of the union of the program order, write-read relation, and the same-transaction relation, i.e., $\mathsf{CO} = (\mathsf{PO} \cup \mathsf{WR} \cup \mathsf{STO})^+$. For readability, we write $ev_1 \rightarrow_{\mathsf{HB}} ev_2$ instead of $(ev_1, ev_2) \in \mathsf{HB}$.

4.2 Program Semantics Under Serializability

The semantics of a program under serializability [37] can be defined using a transition system where the configurations keep a single shared-variable valuation (accessed by all processes) with the standard interpretation of read or write statements. Each transaction executes in isolation. Alternatively, the serializability semantics can be defined as a restriction of $[\mathcal{P}]_{\mathsf{X}}$, $\mathsf{X} \in \{\mathtt{CCv}, \mathtt{CM}, \mathtt{CC}\}$, to the set of executions where each transaction is *immediately* delivered to all processes, i.e., each event $\mathsf{end}(p,t)$ is immediately followed by all $\mathsf{del}(p',t)$ with $p' \neq p$. Such executions are called *serializable* and the set of serializable executions of a program $\mathcal{P}$ is denoted by $\mathbb{E}\mathsf{x}_{\mathtt{SER}}(\mathcal{P})$. The latter definition is easier to reason about when relating executions under causal consistency and serializability, respectively.

A trace *tr* is called *serializable* if it is the trace of a serializable execution. Let $\mathbb{T}\mathsf{r}_{\mathtt{SER}}(\mathcal{P})$ denote the set of serializable traces. Given a serializable trace $tr = (\tau, \mathsf{PO}, \mathsf{WR}, \mathsf{WW}, \mathsf{RW}, \mathsf{STO})$ we have that every event $\mathsf{isu}(p,t)$ in τ is immediately followed by all $\mathsf{del}(p',t)$ with $p' \neq p$. For simplicity, we write τ as a sequence of "atomic macro-events" (p,t) where (p,t) denotes a sequence $\mathsf{isu}(p,t) \cdot \mathsf{del}(p_1,t) \cdot \ldots \cdot \mathsf{del}(p_n,t)$ for some $p \in \mathbb{P} = \{p, p_1, \ldots, p_n\}$. We say that t is "atomic". In Fig. 3, $t2$ is atomic and we can use $(p2, t3)$ instead of $\mathsf{isu}(p2,t3)\mathsf{del}(p1,t3)$.

Since multiple executions may have the same trace, it is possible that an execution ρ produced by a variation of causal consistency has a serializable trace $\mathsf{tr}(\rho)$ even though

```
    p1                       p2
t1 [r1 = x     //0  ||  t2 [r2 = x    //0
    x  = r1 + 1]            x = r2 + 1]
```

(a) Lost Update (LU).

```
    p1                     p2
t1 [x = 1]          ||  t3 [y = 1]
t2 [r1 = y] //0         t4 [r2 = x] //0
```

(b) Store Buffering (SB).

```
[a = 1      if (a == 1)
 z = 1        [x = 2
 x = 1  ||     r1 = z  //0
 y = 1]        r2 = y  //1
               r3 = x] //2
```

(c) Without transactions, non-robust against CCv.

```
[a = 1            if (a == 1)
 x = 1        ||    [x = 2
 r1 = x] //2         r2 = x] //1
```

(d) Without transactions, non-robust only against CM.

```
if ( * )       if ( * )
  [x = 1]   ||   [x = 2]
else           else
  [r1 = x]       [r2 = x]
```

(e) Robust against both CM and CCv.

```
              [r2 = x
[x = 1]   ||   if (r2 == 1)
[r1 = y]        y = 1]
```

(f) Robust against both CM and CCv.

Figure 4 (Non-)robust programs. For non-robust programs, the read instructions are commented with the values they return in robustness violations. The condition of `if-else` is checked inside a transaction whose demarcation is omitted for readability (∗ denotes non-deterministic choice).

$\mathsf{end}(p,t)$ actions may not be immediately followed by $\mathsf{del}(p',t)$ actions. However, ρ would be equivalent, up to reordering of "independent" (or commutative) transitions, to a serializable execution. The happens-before relation between events is extended to transactions as follows: a transaction t_1 *happens-before* another transaction $t_2 \neq t_1$ if the trace tr contains an event of transaction t_1 which happens-before an event of t_2. The happens-before relation between transactions is denoted by HB_t and called *transactional happens-before* (an example is given on the right of Fig. 3). The following result characterizes serializable traces.

▶ **Theorem 3** ([2, 41]). *A trace tr is serializable iff HB_t is acyclic.*

4.3 Robustness Problem

We consider the problem of checking whether the causally-consistent semantics of a program produces the same set of traces as the serializability semantics.

▶ **Definition 4.** *A program $\mathcal{P}$ is called* robust *against a semantics $X \in \{\mathtt{CCv}, \mathtt{CM}, \mathtt{CC}\}$ iff $\mathbb{T}r_X(\mathcal{P}) = \mathbb{T}r_{\mathtt{SER}}(\mathcal{P})$.*

Since $\mathbb{T}r_{\mathtt{SER}}(\mathcal{P}) \subseteq \mathbb{T}r_{\mathsf{X}}(\mathcal{P})$, the problem of checking robustness of a program $\mathcal{P}$ against a semantics X boils down to checking whether there exists a trace $tr \in \mathbb{T}r_{\mathsf{X}}(\mathcal{P}) \setminus \mathbb{T}r_{\mathtt{SER}}(\mathcal{P})$. We call tr a *robustness violation* (or *violation*, for short). By Theorem 3, HB_t of tr is cyclic.

We discuss several examples of programs which are (non-) robust against both CM and CCv or only one of them. Fig. 4a and Fig. 4b show examples of programs that are *not* robust against both CM and CCv, which have also been discussed in the literature on weak memory models, e.g. [6]. The execution of Lost Update under both CM and CCv allows that the two reads of x in transactions $t1$ and $t2$ return 0 although this cannot happen under serializability. Also, executing Store Buffering under both CM and CCv allows that the reads of x and y return 0 although this would not be possible under serializability. These values are possible because the first transaction in each of the processes may not be delivered to the other process.

Assuming for the moment that each instruction in Fig. 4c and Fig. 4d forms a different transaction, the values we give in comments show that the program in Fig. 4c, resp., Fig. 4d, is not robust against CCv, resp., CM. The values in Fig. 4c are possible assuming that the timestamp of the transaction `[x = 1]` is smaller than the timestamp of `[x = 2]` (which means that if the former is delivered after the second process executes `[x = 2]`, then it will be discarded). Moreover, enlarging the transactions as shown in Fig. 4c, the program becomes robust against `CCv`. The values in Fig. 4d are possible under `CM` because different processes do not need to agree on the order in which to apply transactions, each process applying the transaction received from the other process last. However, under `CCv` this behavior is not possible, the program being actually robust against `CCv`. As in the previous case, enlarging the transactions as shown in the figure leads to a robust program against `CM`.

We end the discussion with several examples of programs that are robust against both `CM` and `CCv`. These are simplified models of real applications reported in [28]. The program in Fig. 4e can be understood as the parallel execution of two processes that either create a new user of some service, represented abstractly as a write on a variable `x` or check its credentials, represented as a read of `x` (the non-deterministic choice abstracts some code that checks whether the user exists). Clearly this program is robust against both `CM` and `CCv` since each process does a single access to the shared variable. Although we considered simple transactions that access a single shared-variable this would hold even for "bigger" transactions that access an arbitrary number of variables. The program in Fig. 4f can be thought of as a process creating a new user of some service and reading some additional data in parallel to a process that updates that data only if the user exists. It is rather easy to see that it is also robust against both `CM` and `CCv`.

5 Robustness Against Causal Consistency

We consider now the issue of checking robustness against a variation of causal consistency, considering first the case of `CCv` and `CM`. The result concerning `CC` is derived from the one concerning `CM`. We show next that a robustness violation should contain at least an issue and a store event of the same transaction that are separated by another event that occurs after the issue and before the store and which is related to both via the happens-before relation. Otherwise, since any two events which are not related by happens-before could be swapped in order to derive an execution with the same trace, every store event could be swapped until it immediately follows the corresponding issue and the execution would be serializable.

▶ **Lemma 5.** *Given a robustness violation τ, there exists a transaction t such that $\tau = \alpha \cdot \mathit{isu}(p,t) \cdot \beta \cdot \mathit{del}(p_0,t) \cdot \gamma$ and an event $a \in \beta$ such that $(\mathit{isu}(p,t), a) \in \mathsf{HB}$ and $(a, \mathit{del}(p_0,t)) \in \mathsf{HB}$.*

The transaction t in the trace τ above is called a *delayed* transaction. The happens-before constraints imply that t belongs to an HB_t cycle.

Next, we show that a program which is not robust against `CCv` or `CM` admits violations of particular shapes, which enables reducing robustness checking to a reachability problem in a program running under serializability (presented in Section 6).

5.1 Robustness Violations under Causal Convergence

Roughly, our characterization of `CCv` robustness violations states that the first delayed transaction (which must exist by Lemma 5) is followed by a possibly-empty sequence of delayed transactions that form a "causality chain", i.e., every new delayed transaction is causally ordered after the previous delayed transaction. Moreover, the issue event of the last delayed transaction happens-before the issue event of another transaction that reads a variable updated by the first delayed transaction (this completes a cycle in HB_t). We say that a sequence of issue events $ev_1 \cdot ev_2 \cdot \ldots ev_n$ forms a *causality chain* when $(ev_i, ev_{i+1}) \in \mathsf{CO}$ for all $1 \leq i \leq n-1$. Also, for simplicity, we use "macro-events" (p,t) even in traces obtained under causal consistency (recall that this notation was introduced to simplify serializable traces),

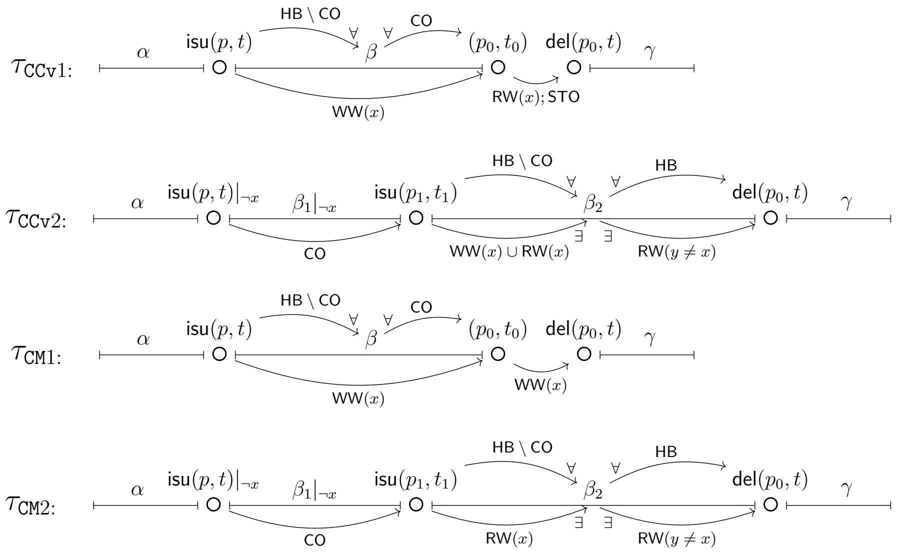


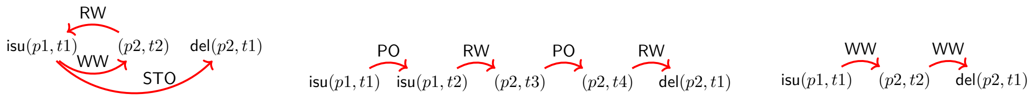
Figure 5 Robustness violation patterns. We use $a \xrightarrow{R\ \forall} \beta$ to denote $\forall\ b \in \beta.\ (a,b) \in R$. We use $\beta_1|_{\neg x}$ to say that all delayed transactions in β_1 do not access x. For violations $\tau_{\texttt{CCv1}}$ and $\tau_{\texttt{CM1}}$, t is the only delayed transaction. For $\tau_{\texttt{CCv2}}$ and $\tau_{\texttt{CM2}}$, all delayed transactions are in $\beta_1|_{\neg x}$ and they form a causality chain that starts at $\mathsf{isu}(p,t)$ and ends at $\mathsf{isu}(p_1,t_1)$.

i.e., we assume that any sequence of events formed of an issue $\mathsf{isu}(p,t)$ followed immediately by all the store events $\mathsf{del}(p',t)$ is replaced by (p,t). Then, all the relations that held between an event ev of such a sequence and another event ev', e.g., $(ev, ev') \in \mathsf{PO}$, are defined to hold as well between the corresponding macro-event (p,t) and ev', e.g, $((p,t), ev') \in \mathsf{PO}$.

▶ **Theorem 6.** *A program $\mathcal{P}$ is* not *robust against* $\texttt{CCv}$ *iff* $\mathbb{T}r(\mathcal{P})_{\texttt{CCv}}$ *contains a trace of type* $\tau_{\texttt{CCv1}} = \alpha \cdot \mathit{isu}(p,t) \cdot \beta \cdot (p_0,t_0) \cdot \mathit{del}(p_0,t) \cdot \gamma$ *or* $\tau_{\texttt{CCv2}} = \alpha \cdot \mathit{isu}(p,t) \cdot \beta_1 \cdot \mathit{isu}(p_1,t_1) \cdot \beta_2 \cdot \mathit{del}(p_0,t) \cdot \gamma$ *that satisfies the properties given in Fig. 5.*

Above, $\tau_{\texttt{CCv1}}$ contains a single delayed transaction while $\tau_{\texttt{CCv2}}$ may contain arbitrarily many delayed transactions. The issue event of the last delayed transactions, i.e., $\mathsf{isu}(p,t)$ in $\tau_{\texttt{CCv1}}$ and $\mathsf{isu}(p_1,t_1)$ in $\tau_{\texttt{CCv2}}$, happens before (p_0,t_0) and some event in β_2, respectively, which read a variable updated by the first delayed transaction. The theorem above allows $\beta_1 = \epsilon$, $\beta_2 = \epsilon$, $\beta = \epsilon$, $\gamma = \epsilon$, $p = p_1$, $t = t_1$, and t_1 to be a read-only transaction. If t_1 is a read-only transaction then $\mathsf{isu}(p_1,t')$ has the same effect as (p_1,t_1) since t_1 does not contain writes. Fig. 6a and Fig. 6b show two violations under $\texttt{CCv}$.

The violation patterns in Theorem 6 characterize *minimal* robustness violations where the measure defined as the sum of the distances (number of events that are causally related to the issue) between issue and store events of the same transaction is minimal. Minimality enforces the constraints stated above. For example, in the context of $\tau_{\texttt{CCv2}}$, the delayed transactions in β_1 cannot create a cycle in the transactional happens-before (otherwise, $\alpha \cdot \mathsf{isu}(p,t) \cdot \beta_1 \cdot \mathsf{del}(p_0,t) \cdot \gamma'$ would be a violation with a smaller measure, which contradicts minimality). Also, if it were to have a delayed transaction t_2 in β_2 (resp., β for $\tau_{\texttt{CCv1}}$), then it is possible to remove some transaction (all its issue and store events) from the original trace and obtain a new violation with a smaller measure. For instance, in the case of β_2, if $t \neq t_1$, then we can safely remove the last delayed transaction (i.e., t_1), that is causally dependent on the first delayed transaction, since all events in $\beta_2 \cdot \mathsf{del}(p_0,t) \cdot \gamma$ neither read from the



(a) Violation of LU program in Fig. 4a.

(b) Violation of SB program in Fig. 4b.

(c) Violation of LU program in Fig. 4a.

Figure 6 (a) A $\tau_{\mathtt{CCv1}}$ violation where $\beta_2 = \epsilon$, $\gamma = \epsilon$, and t and t_0 correspond to $t1$ and $t2$. (b) A $\tau_{\mathtt{CCv2}}$ (resp., $\tau_{\mathtt{CM2}}$) violation where t and t_1 correspond to $t1$ and $t2$. t_1 is a read-only transaction. Also, $\beta_1 = \epsilon$, $\beta_2 = (p2,t3)\cdot(p2,t4)$, $\gamma = \epsilon$, such that $(\mathsf{isu}(p1,t2),(p2,t3)) \in \mathsf{RW}(y)$ and $((p2,t4),\mathsf{del}(p2,t1)) \in \mathsf{RW}(x)$. (c) A $\tau_{\mathtt{CM1}}$ violation with $\beta_2 = \gamma = \epsilon$, and t and t_0 correspond to $t1$ and $t2$. In all traces, we show only the relations that are part of the happens-before cycle.

writes of t_1 nor are issued by the same process as t_1. The resulting trace is still a robustness violation (because of the HB_t cycle involving t_2) but with a smaller measure. Note that all processes that delayed transactions, stop executing new transactions in β_2 (resp., β) because of the relation $\mathsf{HB} \setminus \mathsf{CO}$, shown in Fig. 5, between the delayed transaction t_1 (resp., t) and events in β_2 (resp., β). This characterization of minimal violations is essential for deriving an optimal reduction of robustness checking to SC reachability (presented in Section 6).

5.2 Robustness Violations under Causal Memory

The characterization of robustness violations under `CM` is at some level similar to that of robustness violations under `CCv`. However, some instance of the violation pattern under `CCv` is not possible under `CM` and `CM` admits some class of violations that is not possible under `CCv`. This reflects the fact that these consistency models are incomparable in general. The following theorem gives the precise characterization. Roughly, a program is not robust if it admits a violation which can either be because of two concurrent transactions that write to the same variable (a write-write race) or because of a restriction of the pattern admitted by `CCv` where the last delayed transaction must be related only by RW in a happens-before path with future transactions. The first pattern is not admitted by `CCv` because the writes to each variable are executed according to the timestamp order.

▶ **Theorem 7.** *A program $\mathcal{P}$ is not robust against* `CM` *iff $\mathbb{T}r(\mathcal{P})_{\mathtt{CM}}$ contains a trace of type $\tau_{\mathtt{CM1}} = \alpha\cdot \mathit{isu}(p,t)\cdot\beta\cdot(p_0,t_0)\cdot \mathit{del}(p_0,t)\cdot\gamma$ or $\tau_{\mathtt{CM2}} = \alpha\cdot\mathit{isu}(p,t)\cdot\beta_1\cdot\mathit{isu}(p_1,t_1)\cdot\beta_2\cdot\mathit{del}(p_0,t)\cdot\gamma$ that satisfies the properties given in Fig. 5.*

The `CM` violation in Fig. 6b (pattern $\tau_{\mathtt{CM2}}$) is a violation under `CCv` as well which corresponds to the pattern $\tau_{\mathtt{CCv2}}$. The detection of the violation pattern $\tau_{\mathtt{CM1}}$ (e.g., Fig. 6c) implies the existence of a write-write race under `CM`. Conversely, if a program has a trace which contains a write-write race under `CM`, then this trace must be a robustness violation since the two transactions, that caused the write-write race, form a cycle in the store order hence a cycle in the transactional happens-before order[2]. Thus, the program is not robust against `CM`. Therefore, a program which is robust against `CM` is also write-write race free under `CM`. Since without write-write races, the `CM` and the `CCv` semantics coincide, we get the following.

▶ **Lemma 8.** *If a program $\mathcal{P}$ is robust against* `CM`*, then $\mathcal{P}$ is robust against* `CCv`*.*

[2] Given two concurrent transactions t_1 and t_2 that write on a common variable x, $\mathsf{isu}(p_1,t_1)$ is in store order before $\mathsf{del}(p_1,t_2)$ and $\mathsf{isu}(p_2,t_2)$ before $\mathsf{del}(p_2,t_1)$.

5.3 Robustness Violations under Weak Causal Consistency

If a program is robust against CM, then it must not contain a write-write race under CM (note that this is not true for CCv). Therefore, by Theorem 2, a program which is robust against CM has the same set of traces under both CM and CC, which implies that it is also robust against CC. Conversely, since CC is weaker than CM (i.e., $\mathbb{T}\mathsf{r}_{\mathtt{CM}}(\mathcal{P}) \subseteq \mathbb{T}\mathsf{r}_{\mathtt{CC}}(\mathcal{P})$ for any $\mathcal{P}$), if a program is robust against CC then it is robust against CM. Thus, we obtain the following.

▶ **Theorem 9.** *A program $\mathcal{P}$ is robust against* CC *iff it is robust against* CM*.*

6 Reduction to SC Reachability

We present a reduction of robustness checking to a reachability problem in a program executing under the serializability semantics. Given a program $\mathcal{P}$ and a semantics $\mathsf{X} \in \{\mathtt{CCv}, \mathtt{CM}, \mathtt{CC}\}$, we define an instrumentation of $\mathcal{P}$ such that $\mathcal{P}$ is not robust against X iff the instrumentation reaches an error state under serializability. The instrumentation uses auxiliary variables to simulate the robustness violations (in particular, the delayed transactions) satisfying the patterns given in Fig. 5. We focus our presentation on the second violation pattern of CCv (similar to the second violation pattern of CM): $\tau_{\mathtt{CCv2}} = \alpha \cdot \mathsf{isu}(p,t) \cdot \beta_1 \cdot \mathsf{isu}(p_1,t_1) \cdot \beta_2 \cdot \mathsf{del}(p_0,t) \cdot \gamma$. We describe the instrumentation only informally, the precise definition can be found in [11].

The process p that delayed the first transaction t is called the *Attacker*. The processes delaying transactions in $\beta_1 \cdot \mathsf{isu}(p_1,t_1)$ are called *Visibility Helpers*. Recall that all the delayed transactions must be causally after $\mathsf{isu}(p,t)$ and causally before $\mathsf{isu}(p_1,t_1)$. The processes that execute transactions in β_2 and contribute to the happens-before path between $\mathsf{isu}(p_1,t_1)$ and $\mathsf{del}(p_0,t)$ are called *Happens-Before Helpers*. A happens-before helper cannot be the attacker or a visibility helper since this would contradict causal delivery (a transaction of a happens-before helper is not delayed, so it is visible immediately to all processes, and it cannot follow a delayed transaction). γ contains the stores of the delayed transactions from $\mathsf{isu}(p,t) \cdot \beta_1 \cdot \mathsf{isu}(p_1,t_1)$. It is important to notice that we may have $t = t_1$. In this case, $\beta_1 = \epsilon$ and the only delayed transaction is t. Also, all delayed transactions in β_1 including t_1 may be issued on the same process as t. In all these cases, the set of Visibility Helpers is empty.

The instrumentation uses two copies of the set of shared variables in the original program. We use primed variables x' to denote the second copy. When a process becomes the attacker or a visibility helper, it will write only to the second copy that is visible only to these processes (and remains invisible to the other processes including the happens-before helpers). The writes made by the other processes including the happens-before helpers are made visible to all processes, i.e., they are applied on both copies of the shared variables.

To establish the causality chains of delayed transactions performed by the attacker and the visibility helpers, we look whether a transaction can extend the causality chain started by the first delayed transaction from the attacker. In order for a transaction to "join" the causality chain, it has to satisfy one of the following conditions:

- the transaction is issued by a process that has already another transaction in the causality chain. Thus, we ensure the continuity of the causality chain through program order.
- the transaction is reading from a variable updated by a previous transaction in the causality chain. Hence, we ensure the continuity of the causality chain through write-read.

We introduce a flag for each shared variable to mark the fact that it was updated by a previous transaction in the causality chain. These flags are used by the instrumentation to establish whether a transaction "joins" a causality chain. Enforcing a happens-before path starting in the last delayed transaction, using transactions of the happens-before helpers, can be done in the similar way. Compared to causality chains, there are two more cases in which a transaction can extend a happens-before path:

- the transaction writes to a variable read by a previous transaction in the happens-before path. Hence, we ensure the continuity of the happens-before path through read-write.
- the transaction writes to a variable updated by a previous transaction in the happens-before path. We ensure the continuity of the happens-before path through write-write.

We extend the set of flags used for causality chains to record if a variable was read or written by a previous transaction in the happens-before path. Overall, the instrumentation uses a flag $x.event$ or $x'.event$ for each (copy of a) shared variable, that records the type of the last access (read or write) to that variable. Initially, these flags and other flags used by the instrumentation as explained below are initialized to null ($\perp$).

In general, whether a process is an attacker, visibility helper, or happens-before helper is not enforced syntactically by the instrumentation, and can vary from execution to execution. The role of a process in an execution is set *non-deterministically* during the execution using some additional process-local flags. Thus, during an execution, each process chooses to set to true at most one of the flags $p.a$, $p.vh$, and $p.hbh$, implying that the process becomes an attacker, visibility helper, or happens-before helper, respectively. At most one process can be an attacker, i.e., set $p.a$ to true. Before a process becomes an attacker or visibility helper it passes though a stage where it executes transactions in the usual way without delaying them.

A process can non-deterministically choose to delay a transaction at which point it sets a *global* flag $a_{\mathsf{tr}_\mathsf{A}}$ to true. During the delayed transaction it chooses randomly a write instruction to a shared variable y and stores the name of this variable in the global variable $a_{\mathsf{st}_\mathsf{A}}$. The values written during delayed transactions are stored in primed variables and visible only to the attacker and the visibility helpers. Each time the attacker writes to a variable z' (the copy of z from the original program) during a delayed transaction, it sets the flag $z'.event$ to st which will allow other processes that read the same variable to join the set of visibility helpers and start delaying their transactions. Once the attacker delays a transaction, it starts reading only from the primed variables (i.e., z').

When $a_{\mathsf{tr}_\mathsf{A}}$ is set to true by the attacker, other processes continue the execution of their original instructions but, whenever they store a value they write it to both the shared variable z and the primed variable z' so it is visible to all processes. When a process chooses non deterministically to join the visibility helpers, it delays all writes (i.e., writes only to primed variables) and reads only from the primed variables.

In order for the attacker or a visibility helper to start the happens-before path, it has to either read or write a shared variable x that was not accessed by a delayed transaction (i.e., $x'.event = \perp$). In this case we set the global flag HB to true to mark the start of the happens-before path and the end of the causality chain, and set the flag $x.event$ to ld. When HB becomes true the attacker and the visibility helpers stop executing new transactions.

Also, when HB becomes true, the remaining processes can choose non-deterministically to join the set of happens-before helpers, i.e., continue the happens-before path created by the existing happens-before helpers, the attacker, or visibility helper. The happens-before helpers continue executing their instructions, until one of them reads from the variable y whose name was stored in $a_{\mathsf{st}_\mathsf{A}}$. This establishes a happens-before path between the last delayed transaction and a "fictitious" store event corresponding to the first delayed transaction that could be executed just after this read of y. The execution doesn't have to contain this store event explicitly since it is always enabled. Therefore, at the end of every transaction, the instrumentation checks whether the transaction reads y. If this is the case, then the execution stops and goes to an error state to indicate that this is a robustness violation.

The role of a process in an execution is chosen non-deterministically at runtime. Therefore, the instrumentation $[\![\mathcal{P}]\!]$ of a program $\mathcal{P}$ is obtained by replacing each instruction $\langle linst \rangle$ with the concatenation of the instrumentations corresponding to the attacker, the visibility helpers, and the happens-before helpers, i.e., $[\![\langle linst \rangle]\!] ::= [\![\langle linst \rangle]\!]_{\mathsf{A}}\ [\![\langle linst \rangle]\!]_{\mathsf{VH}}\ [\![\langle linst \rangle]\!]_{\mathsf{HbH}}$. The following theorem states the correctness of the instrumentation.

▶ **Theorem 10.** *A program $\mathcal{P}$ is not robust against* `CCv` *iff $[\![\mathcal{P}]\!]$ reaches the error state.*

A similar instrumentation can be defined for the other variations of causal consistency, i.e., causal memory (`CM`) and weak causal consistency (`CC`).

The following result states the complexity of checking robustness for finite-state programs [3] against one of the three variations of causal consistency considered in this work (we use causal consistency as a generic name to refer to all of them). It is a direct consequence of Theorem 10 and of previous results concerning the reachability problem in concurrent programs running over SC, with a fixed [27] or parametric number of processes [40].

▶ **Corollary 11.** *Checking robustness of finite-state programs against causal consistency is PSPACE-complete when the number of processes is fixed and EXPSPACE-complete, otherwise.*

7 Related Work

Causal consistency is one of the oldest consistency models for distributed systems [30]. Formal definitions of several variants of causal consistency, suitable for different types of applications, have been introduced recently [19, 18, 38, 14]. The definitions in this paper are inspired from these works and coincide with those given in [14]. In that paper, the authors address the decidability and the complexity of verifying that an implementation of a storage system is causally consistent (i.e., all its computations, for every client, are causally consistent).

While our paper focuses on *trace-based* robustness, *state-based robustness* requires that a program is robust if the set of all its reachable states under the weak semantics is the same as its set of reachable states under the strong semantics. While state-robustness is the necessary and sufficient concept for preserving state-invariants, its verification, which amounts in computing the set of reachable states under the weak semantics, is in general a hard problem. The decidability and the complexity of this problem has been investigated in the context of relaxed memory models such as TSO and Power, and it has been shown that it is either decidable but highly complex (non-primitive recursive), or undecidable [8, 9]. As far as we know, the decidability and complexity of this problem has not been investigated for causal consistency. Automatic procedures for approximate reachability/invariant checking have been proposed using either abstractions or bounded analyses, e.g., [10, 5, 21, 1]. Proof methods have also been developed for verifying invariants in the context of weakly consistent models such as [29, 26, 36, 4]. These methods, however, do not provide decision procedures.

Decidability and complexity of trace-based robustness has been investigated for the TSO and Power memory models [15, 13, 22]. The work we present in this paper borrows the idea of using minimal violation characterizations for building an instrumentation allowing to obtain a reduction of the robustness checking problem to the reachability checking problem over SC. However, applying this approach to the case of causal consistency is not straightforward and requires different proof techniques. Dealing with causal consistency is far more tricky and difficult than dealing with TSO, and requires coming up with radically different arguments

[3] That is, programs where the number of variables and the data domain are bounded.

and proofs, for (1) characterizing in a finite manner the set of violations, (2) showing that this characterization is sound and complete, and (3) using effectively this characterization in the definition of the reduction to the reachability problem.

As far as we know, our work is the first one that establishes results on the decidability and complexity issues of the robustness problem in the context of causal consistency, and taking into account transactions. The existing work on the verification of robustness for distributed systems consider essentially trace-based concepts of robustness and provide either over- or under-approximate analyses for checking it. The static analyses proposed in [12, 16, 17, 20] are based on computing an abstraction of the set of computations, which is used in searching for robustness violations. These approaches may return false alarms due to the abstractions they consider. In particular, [12] shows that a trace under causal convergence is not admitted by the serializability semantics iff it contains a (transactional) happens-before cycle with a RW dependency, and another RW or WW dependency. This characterization alone is not sufficient to prove the reduction in Theorem 10, which requires a finer characterization of robustness violations. A sound (but not complete) bounded analysis for detecting robustness violation is proposed in [35]. Our approach is technically different, is precise, and provides a decision procedure for checking robustness when the program is finite-state.

References

1 Parosh Aziz Abdulla, Mohamed Faouzi Atig, Ahmed Bouajjani, and Tuan Phong Ngo. Context-Bounded Analysis for POWER. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part II*, volume 10206 of *Lecture Notes in Computer Science*, pages 56–74, 2017.

2 Atul Adya. *Weak consistency: A generalized theory and optimistic implementations for distributed transactions.* PhD thesis, MIT, 1999.

3 Mustaque Ahamad, Gil Neiger, James E. Burns, Prince Kohli, and Phillip W. Hutto. Causal Memory: Definitions, Implementation, and Programming. *Distributed Computing*, 9(1):37–49, 1995.

4 Jade Alglave and Patrick Cousot. Ogre and Pythia: an invariance proof method for weak consistency models. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 3–18. ACM, 2017.

5 Jade Alglave, Daniel Kroening, and Michael Tautschnig. Partial Orders for Efficient Bounded Model Checking of Concurrent Software. 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 141–157. Springer, 2013.

6 Jade Alglave, Luc Maranget, and Michael Tautschnig. Herding Cats: Modelling, Simulation, Testing, and Data Mining for Weak Memory. *ACM Trans. Program. Lang. Syst.*, 36(2):7:1–7:74, 2014.

7 Sérgio Almeida, João Leitão, and Luís E. T. Rodrigues. ChainReaction: a causal+ consistent datastore based on chain replication. In Zdenek Hanzálek, Hermann Härtig, Miguel Castro, and M. Frans Kaashoek, editors, *Eighth Eurosys Conference 2013, EuroSys '13, Prague, Czech Republic, April 14-17, 2013*, pages 85–98. ACM, 2013.

8 Mohamed Faouzi Atig, Ahmed Bouajjani, Sebastian Burckhardt, and Madanlal Musuvathi. On the verification problem for weak memory models. In Manuel V. Hermenegildo and Jens Palsberg, editors, *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2010, Madrid, Spain, January 17-23, 2010*, pages 7–18. ACM, 2010.

9 Mohamed Faouzi Atig, Ahmed Bouajjani, Sebastian Burckhardt, and Madanlal Musuvathi. What's Decidable about Weak Memory Models? 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 26–46. Springer, 2012.

10 Mohamed Faouzi Atig, Ahmed Bouajjani, and Gennaro Parlato. Getting Rid of Store-Buffers in TSO Analysis. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*, pages 99–115. Springer, 2011.

11 Sidi Mohamed Beillahi, Ahmed Bouajjani, and Constantin Enea. Robustness Against Transactional Causal Consistency. *CoRR*, 2019.

12 Giovanni Bernardi and Alexey Gotsman. Robustness against Consistency Models with Atomic Visibility. 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 7:1–7:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

13 Ahmed Bouajjani, Egor Derevenetc, and Roland Meyer. Checking and Enforcing Robustness against TSO. In Matthias Felleisen and Philippa Gardner, editors, *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7792 of *Lecture Notes in Computer Science*, pages 533–553. Springer, 2013.

14 Ahmed Bouajjani, Constantin Enea, Rachid Guerraoui, and Jad Hamza. On verifying causal consistency. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 626–638. ACM, 2017.

15 Ahmed Bouajjani, Roland Meyer, and Eike Möhlmann. Deciding Robustness against Total Store Ordering. In Luca Aceto, Monika Henzinger, and Jirí Sgall, editors, *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II*, volume 6756 of *Lecture Notes in Computer Science*, pages 428–440. Springer, 2011.

16 Lucas Brutschy, Dimitar Dimitrov, Peter Müller, and Martin T. Vechev. Serializability for eventual consistency: criterion, analysis, and applications. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL 2017, Paris, France, January 18-20, 2017*, pages 458–472. ACM, 2017.

17 Lucas Brutschy, Dimitar Dimitrov, Peter Müller, and Martin T. Vechev. Static serializability analysis for causal consistency. 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 90–104. ACM, 2018.

18 Sebastian Burckhardt. Principles of Eventual Consistency. *Foundations and Trends in Programming Languages*, 1(1-2):1–150, 2014.

19 Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski. Replicated data types: specification, verification, optimality. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 271–284. ACM, 2014.

20 Andrea Cerone and Alexey Gotsman. Analysing Snapshot Isolation. *J. ACM*, 65(2):11:1–11:41, 2018.

21 Andrei Marian Dan, Yuri Meshman, Martin T. Vechev, and Eran Yahav. Effective abstractions for verification under relaxed memory models. *Computer Languages, Systems & Structures*, 47:62–76, 2017.

22 Egor Derevenetc and Roland Meyer. Robustness against Power is PSpace-complete. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 158–170. Springer, 2014.

23 Jiaqing Du, Sameh Elnikety, Amitabha Roy, and Willy Zwaenepoel. Orbe: scalable causal consistency using dependency matrices and physical clocks. In Guy M. Lohman, editor, *ACM Symposium on Cloud Computing, SOCC '13, Santa Clara, CA, USA, October 1-3, 2013*, pages 11:1–11:14. ACM, 2013.

24 Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM*, 32(2):374–382, 1985.

25 Seth Gilbert and Nancy A. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, 2002.

26 Alexey Gotsman, Hongseok Yang, Carla Ferreira, Mahsa Najafzadeh, and Marc Shapiro. 'Cause I'm strong enough: reasoning about consistency choices in distributed systems. In Rastislav Bodík and Rupak Majumdar, editors, *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 371–384. ACM, 2016.

27 Dexter Kozen. Lower Bounds for Natural Proof Systems. In *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, pages 254–266. IEEE Computer Society, 1977.

28 Arthur Kurath. Analyzing Serializability of Cassandra Applications. Master's thesis, ETH Zurich, Switzerland, 2017.

29 Ori Lahav and Viktor Vafeiadis. Owicki-Gries Reasoning for Weak Memory Models. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 311–323. Springer, 2015.

30 Leslie Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21(7):558–565, 1978.

31 Leslie Lamport. How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. *IEEE Trans. Computers*, 28(9):690–691, 1979.

32 Richard J Lipton and Jonathan S Sandberg. PRAM: A scalable shared memory. Technical Report TR-180-88, Princeton University, Department of Computer Science, August 1988.

33 Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In Ted Wobber and Peter Druschel, editors, *Proceedings of the 23rd ACM Symposium on Operating Systems Principles 2011, SOSP 2011, Cascais, Portugal, October 23-26, 2011*, pages 401–416. ACM, 2011.

34 Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. Stronger Semantics for Low-Latency Geo-Replicated Storage. In Nick Feamster and Jeffrey C. Mogul, editors, *Proceedings of the 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2013, Lombard, IL, USA, April 2-5, 2013*, pages 313–328. USENIX Association, 2013.

35 Kartik Nagar and Suresh Jagannathan. Automatic Detection of Serializability Violations Under Weak Consistency. In *29th Intern. Conf. on Concurrency Theory (CONCUR'18)*, September 2018. to appear.

36 Mahsa Najafzadeh, Alexey Gotsman, Hongseok Yang, Carla Ferreira, and Marc Shapiro. The CISE tool: proving weakly-consistent applications correct. In Peter Alvaro and Alysson Bessani, editors, *Proceedings of the 2nd Workshop on the Principles and Practice of Consistency for Distributed Data, PaPoC@EuroSys 2016, London, United Kingdom, April 18, 2016*, pages 2:1–2:3. ACM, 2016.

37 Christos H. Papadimitriou. The serializability of concurrent database updates. *J. ACM*, 26(4):631–653, 1979.

38 Matthieu Perrin, Achour Mostéfaoui, and Claude Jard. Causal consistency: beyond memory. In Rafael Asenjo and Tim Harris, editors, *Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2016, Barcelona, Spain, March 12-16, 2016*, pages 26:1–26:12. ACM, 2016.

39 Nuno M. Preguiça, Marek Zawirski, Annette Bieniusa, Sérgio Duarte, Valter Balegas, Carlos Baquero, and Marc Shapiro. SwiftCloud: Fault-Tolerant Geo-Replication Integrated all the Way to the Client Machine. In *33rd IEEE International Symposium on Reliable Distributed Systems Workshops, SRDS Workshops 2014, Nara, Japan, October 6-9, 2014*, pages 30–33. IEEE Computer Society, 2014.

40 Charles Rackoff. The Covering and Boundedness Problems for Vector Addition Systems. *Theor. Comput. Sci.*, 6:223–231, 1978.

41 Dennis E. Shasha and Marc Snir. Efficient and Correct Execution of Parallel Programs that Share Memory. *ACM Trans. Program. Lang. Syst.*, 10(2):282–312, 1988.

Expressive Power of Broadcast Consensus Protocols

Michael Blondin

Département d'informatique, Université de Sherbrooke, Sherbrooke, Canada
michael.blondin@usherbrooke.ca

Javier Esparza

Fakultät für Informatik, Technische Universität München, Garching bei München, Germany
esparza@in.tum.de

Stefan Jaax

Fakultät für Informatik, Technische Universität München, Garching bei München, Germany
jaax@in.tum.de

Abstract

Population protocols are a formal model of computation by identical, anonymous mobile agents interacting in pairs. Their computational power is rather limited: Angluin et al. have shown that they can only compute the predicates over $\mathbb{N}^k$ expressible in Presburger arithmetic. For this reason, several extensions of the model have been proposed, including the addition of devices called cover-time services, absence detectors, and clocks. All these extensions increase the expressive power to the class of predicates over $\mathbb{N}^k$ lying in the complexity class NL when the input is given in unary. However, these devices are difficult to implement, since they require that an agent atomically receives messages from *all* other agents in a population of unknown size; moreover, the agent must *know* that they have all been received. Inspired by the work of the verification community on Emerson and Namjoshi's broadcast protocols, we show that NL-power is also achieved by extending population protocols with reliable broadcasts, a simpler, standard communication primitive.

2012 ACM Subject Classification Theory of computation → Distributed computing models; Theory of computation → Complexity classes; Theory of computation → Automata over infinite objects

Keywords and phrases population protocols, complexity theory, counter machines, distributed computing

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.31

Related Version `https://arxiv.org/abs/1902.01668`

Funding *Michael Blondin*: Supported by the Fonds de recherche du Québec – Nature et technologies (FRQNT), by a Quebec–Bavaria project funded by the Fonds de recherche du Québec (FRQ), and by the Natural Sciences and Engineering Research Council of Canada (NSERC).
Javier Esparza: Supported by an ERC Advanced Grant (787367: PaVeS).
Stefan Jaax: Supported by an ERC Advanced Grant (787367: PaVeS).

Acknowledgements Part of this work was realized while Stefan Jaax was visiting the Université de Sherbrooke. We warmly thank the anonymous reviewers for their helpful comments and suggestions.

1 Introduction

Population protocols are a theoretical model for the study of ad hoc networks of tiny computing devices without any infrastructure [5, 6], intensely investigated in recent years (see e.g. [2, 3, 4, 14]). The model postulates a "soup" of indistinguishable agents that behave identically, and only have a fixed number of bits of memory, i.e., a finite number of local states. Agents repeatedly interact in pairs, changing their states according to a joint transition function. A global fairness condition ensures that every finite sequence of interactions that

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 31; pp. 31:1–31:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

becomes enabled infinitely often is also executed infinitely often. The purpose of a population protocol is to allow agents to collectively compute some information about their initial configuration, defined as the function that assigns to each local state the number of agents that initially occupy it. For example, assume that initially each agent picks a boolean value by choosing, say, q_0 or q_1 as its initial state. The many *majority protocols* described in the literature allow the agents to eventually reach a stable consensus on the value chosen by a majority of the agents. More formally, let x_0 and x_1 denote the initial numbers of agents in states q_0 and q_1; majority protocols compute the predicate $\varphi\colon \mathbb{N} \times \mathbb{N} \to \{0, 1\}$ given by $\varphi(x_0, x_1) = (x_1 \geq x_0)$. Throughout the paper, we use the term "predicate" as an abbreviation for "function from $\mathbb{N}^k$ to $\{0, 1\}$ for some k".

In a seminal paper, Angluin et al. proved that population protocols compute exactly the predicates expressible in Presburger arithmetic [6, 7]. Thus, for example, agents can decide if they are at least a certain number, if at least 2/3 of them voted the same way, or, more generally, if the vector $(x_1, x_2, \ldots, x_n)$ representing the number of agents that picked option $1, 2, \ldots, n$ in an election with n choices is a solution of a system of linear inequalities. On the other hand, they cannot decide if they are a square or a prime number, or if the product of the number of votes for options 1 and 2 exceeds the number of votes for option 3. Much work has been devoted to designing more powerful formalisms and analyzing their expressive power. In particular, population protocols have recently been extended with capabilities allowing an agent to obtain global information about the current configuration, which we proceed to describe.

In [22], Michail and Spirakis extend the population protocol model with *absence detectors*, by means of which an agent knows, for every state, whether the state is currently populated or not. Further, they implement absence detectors by a weaker object called a *cover-time service*, which allows an agent to deduce if it has interacted with every other agent in the system. They prove that protocols with cover-time can compute all predicates in $\mathsf{DSPACE}(\log n)$ and can only compute predicates in $\mathsf{NSPACE}(\log n) = \mathsf{NL}$, where n is the number of agents[1].

In [8], Aspnes observes that cover-time services are a kind of internal clock mechanism, and introduces clocked population protocols. Clocked protocols have a clock oracle that signals to one or more agents that the population has reached a bottom strongly connected component of the configuration graph, again an item of global information. Aspnes shows that clocked protocols can compute exactly the predicates in NL.

Absence detectors, cover-time services, and clocked protocols are difficult to implement, since they require that an agent reliably receives information from *all* other agents; moreover, the agent needs to *know* that it has already received messages from all other agents before making a move, which is particularly difficult because agents are assumed to have no identities and to ignore the size of the population. In this paper, we propose a much simpler extension (from an implementation point of view): We allow agents to perform reliable broadcasts, a standard operation in concurrency and distributed computing. We are inspired by the broadcast protocol model introduced by Emerson and Namjoshi in [15] to describe bus-based hardware protocols. The model has been used and further studied in many other contributions, e.g. [16, 18, 12, 24, 9]. In broadcast protocols, agents can perform binary interactions, as in the population protocol model, but, additionally, an agent can also broadcast a signal to all other agents, which are guaranteed to react to it. Broadcast protocols are rather simple to

[1] Observe that, for example, n agents can decide whether n is prime. Indeed, a Turing machine can decide if n is a prime number in $\Theta(\log n)$ space by going through all numbers from 2 to $n - 1$, and checking for each of them if they divide n.

implement with current technology on mobile agents moving in a limited area. Broadcasts also appear in biological systems. For example, Uhlendorf et al. describe a system in which a controller adds a sugar or saline solution to a population of yeasts, to which all the yeasts react [27]. An idealized model of the system, which is essentially a broadcast protocol, has been analyzed by Bertrand et al. in [9].

In this paper, we show that population protocols with reliable broadcasts also compute *precisely* the predicates in NL, and are therefore as powerful as absence detectors or clocks. To prove this result, we first define the notion of *silent semi-computation*, a weaker notion than standard computation, and prove that broadcast protocols silently semi-compute all protocols in NL. This result makes crucial use of the ability of broadcast protocols to "restart" the whole population nondeterministically whenever something bad or unexpected is detected. We then prove that silent semi-computability and computability coincide for the class NL.

In a second contribution, we explore in more detail the minimal requirements for achieving NL power. On the one hand, we show that it is enough to allow *a single* agent to broadcast *a single* signal. On the other hand, we prove that the addition of a reset, which causes all agents to return to their initial states, does not increase the power of population protocols.

2 Preliminaries

Multisets. A *multiset* over a finite set E is a mapping $M\colon E \to \mathbb{N}$. The set of all multisets over E is denoted $\mathbb{N}^E$. For every $e \in E$, $M(e)$ denotes the number of occurrences of e in M. We sometimes denote multisets using a set-like notation, e.g. $\wr f, g, g \wr$ is the multiset M such that $M(f) = 1$, $M(g) = 2$ and $M(e) = 0$ for every $e \in E \setminus \{f, g\}$. Addition and comparison are extended to multisets componentwise, i.e. $(M + M')(e) \overset{\text{def}}{=} M(e) + M'(e)$ for every $e \in E$, and $M \leq M' \overset{\text{def}}{\iff} M(e) \leq M'(e)$ for every $e \in E$. We define multiset difference as $(M \ominus M')(e) \overset{\text{def}}{=} \max(M(e) - M'(e), 0)$ for every $e \in E$. The empty multiset is denoted $\mathbf{0}$ and, for every $e \in E$, we write $\boldsymbol{e} \overset{\text{def}}{=} \wr e \wr$. Finally, we define the *support* and *size* of $M \in \mathbb{N}^E$ respectively as $[\![M]\!] \overset{\text{def}}{=} \{e \in E : M(e) > 0\}$ and $|M| \overset{\text{def}}{=} \sum_{e \in E} M(e)$.

Population protocols. A *population* over a finite set E is a multiset $P \in \mathbb{N}^E$ such that $|P| \geq 2$. The set of all populations over E is denoted by $\mathrm{Pop}(E)$. A *population protocol with leaders* (population protocol for short) is a tuple $\mathcal{P} = (Q, R, \Sigma, L, I, O)$ where:

- Q is a non-empty finite set of *states*,
- $R \subseteq (Q \times Q) \times (Q \times Q)$ is a set of *rendez-vous transitions*,
- Σ is a non-empty finite *input alphabet*,
- $I\colon \Sigma \to Q$ is the *input function* mapping input symbols to states,
- $L \in \mathbb{N}^Q$ is the multiset of *leaders*, and
- $O\colon Q \to \{0, 1\}$ is the *output function* mapping states to boolean values.

Following the standard convention, we call elements of $\mathrm{Pop}(Q)$ *configurations*. Intuitively, a configuration C describes a collection of identical finite-state *agents* with Q as set of states, containing $C(q)$ agents in state q for every $q \in Q$, and at least two agents in total.

We write $(p, q) \mapsto (p', q')$ to denote that $(p, q, p', q') \in R$. The relation $\mathrm{Step}\colon \mathrm{Pop}(Q) \to \mathrm{Pop}(Q)$ is defined by: $(C, C') \in \mathrm{Step}$ iff there exists $(p, q, p', q') \in R$ such that $C \geq \wr p, q \wr$ and $C' = C \ominus \wr p, q \wr + \wr p', q' \wr$. We write $C \to C'$ if $(C, C') \in \mathrm{Step}$, and $C \xrightarrow{*} C'$ if $(C, C') \in \mathrm{Step}^*$, the reflexive and transitive closure of Step. If $C \xrightarrow{*} C'$, then we say that C' is *reachable* from C. An *execution* is an infinite sequence of configurations $C_0 C_1 \cdots$ such that $C_i \to C_{i+1}$ for every $i \in \mathbb{N}$. An execution $C_0 C_1 \cdots$ is *fair* if for every step $C \to C'$ the following holds: if $C_i = C$ for infinitely many indices $i \in \mathbb{N}$, then $C_j = C'$ for infinitely many indices $j \in \mathbb{N}$.

We now explain the roles of the input function I and the multiset L of leaders. The elements of $\mathrm{Pop}(\Sigma)$ are called *inputs*. For every input $X \in \mathrm{Pop}(\Sigma)$, let $I(X) \in \mathrm{Pop}(Q)$ denote the configuration defined by

$$I(X)(q) \overset{\text{def}}{=} \sum_{\{\sigma\in\Sigma : I(\sigma)=q\}} X(\sigma) \qquad\qquad \text{for every } q \in Q.$$

A configuration C is *initial* if $C = I(X) + L$ for some input X. Intuitively, the agents of $I(X)$ encode the input, while those of L are a fixed number of agents, traditionally called leaders, that perform the computation together with the agents of $I(X)$.

Predicate computed by a protocol. If $O(p) = O(q)$ for every $p, q \in [\![C]\!]$, then C is a *consensus configuration*, and $O(C)$ denotes the unique output of the states in $[\![C]\!]$. We say that a consensus configuration C is a *b-consensus* if $O(C) = b$. An execution $C_0C_1\cdots$ *stabilizes* to $b \in \{0,1\}$ if there exists $n \in \mathbb{N}$ such that C_i is a b-consensus for every $i \geq n$.

A protocol $\mathcal{P}$ over an input alphabet Σ *computes* a predicate $\varphi\colon \mathrm{Pop}(\Sigma) \to \{0,1\}$ if for every input $X \in \mathrm{Pop}(\Sigma)$, every fair execution of $\mathcal{P}$ starting at the initial configuration $I(X) + L$ stabilizes to $\varphi(X)$.

Throughout the paper, we assume $\Sigma = \{A_1, \ldots, A_k\}$ for some $k > 0$. Abusing language, we identify population $M \in \mathrm{Pop}(\Sigma)$ to vector $\boldsymbol{\alpha} = (M(A_1), \ldots, M(A_k))$, and say that $\mathcal{P}$ computes a *predicate* $\varphi\colon \mathbb{N}^k \to \{0,1\}$ *of arity* k. In the rest of the paper, the term "predicate" is used with the meaning "function from $\mathbb{N}^k$ to $\{0,1\}$". It is known that:

▶ **Theorem 1** ([7]). *Population protocols compute exactly the predicates expressible in Presburger arithmetic, i.e. the first-order theory of the natural numbers with addition.*

3 Broadcast consensus protocols

Broadcast protocols were introduced by Emerson and Namjoshi in [15] as a formal model of bus-based hardware protocols, such as those for cache coherency. The model has also been applied to the verification of multithreaded programs [12], and to idealized modeling of control problems for living organisms [27, 9]. Its theory has been further studied in [16, 18, 24].

Agents of broadcast protocols can communicate in pairs, as in population protocols, and, additionally, they can also communicate by means of a reliable broadcast. An agent can broadcast a signal to all other agents, which after receiving the signal move to a new state. Broadcasts are routinely used in wireless ad-hoc and sensor networks (see e.g. [1, 28]), and so they are easy to implement on the same kind of systems targeted by population protocols. They can also model idealized versions of communication in natural computing. For example, in [9] they are used to model "communication" in which an experimenter "broadcasts" a signal to a colony of yeasts by increasing the concentration of a nutrient in a solution.

We introduce broadcast consensus protocols, i.e., broadcast protocols whose goal is to compute a predicate in the computation-by-consensus paradigm.

▶ **Definition 2.** *A broadcast consensus protocol is a tuple $\mathcal{P} = (Q, R, B, \Sigma, L, I, O)$, where all components but B are defined as for population protocols, and B is a set of* broadcast transitions*. A* broadcast transition *is a triple (q, r, f) where $q, r \in Q$ and $f\colon Q \to Q$ is a* transfer function.

The relation Step $\subseteq \mathrm{Pop}(Q) \times \mathrm{Pop}(Q)$ *of $\mathcal{P}$ is defined as follows. A pair (C, C') of configurations belongs to* Step *iff*

- *there exists* $(p, q) \mapsto (p', q') \in R$ *such that* $C \geq \llparenthesis p, q \rrparenthesis$ *and* $C' = C \ominus \llparenthesis p, q \rrparenthesis + \llparenthesis p', q' \rrparenthesis$*; or*
- *there exists a transition* $(q, r, f) \in B$ *such that* $C(q) \geq 1$ *and* C' *is the configuration computed from* C *in the following three steps:*

$$C_1 = C \ominus \llparenthesis q \rrparenthesis, \tag{1}$$

$$C_2(q') = \sum_{r' \in f^{-1}(q')} C_1(r') \qquad \textit{for every } q' \in Q, \tag{2}$$

$$C' = C_2 + \llparenthesis r \rrparenthesis. \tag{3}$$

Intuitively, (1)–(3) is interpreted as follows: (1) an agent at state q broadcasts a signal and leaves q, yielding C_1; (2) all other agents receive the signal and move to the states indicated by the function f, yielding C_2; and (3) the broadcasting agent enters state r, yielding C'. Correspondingly, instead of (q, r, f) we use $q \mapsto r;\ f$ as notation for a broadcast transition.

Beyond Presburger arithmetic. As a first illustration of the power of broadcast protocols, we show that their expressive power goes beyond Presburger arithmetic, and so beyond the power of population protocols. We present a broadcast consensus protocol for the predicate φ, defined as $\varphi(x) = 1$ iff $x > 1$ and x is a power of two. For readability, we use the notation $q \mapsto q'; [q_1 \mapsto q'_1, \ldots, q_n \mapsto q'_n]$ for a broadcast transition, where $f(q_i) = q'_i$ and where transfers of the form $q_i \mapsto q_i$ may be omitted.

Let $\mathcal{P} = (Q, R, B, \Sigma, L, I, O)$ be the broadcast consensus protocol where $Q \overset{\text{def}}{=} \{x, \overline{x}, \tilde{x}, 0, 1, \bot\}$, $\Sigma \overset{\text{def}}{=} \{x\}$, $I \overset{\text{def}}{=} x \mapsto x$, $L \overset{\text{def}}{=} \mathbf{0}$, $O(q) = 1 \overset{\text{def}}{\iff} q = 1$, and R and B are defined as follows:

- R contains the rendez-vous transition $s\colon (x, x) \mapsto (\overline{x}, 0)$;
- B contains the broadcast transitions $r\colon \bot \mapsto x; [q \mapsto x : q \in Q]$ and

$$\overline{s}\colon \overline{x} \mapsto x; \begin{bmatrix} x \mapsto \bot \\ \overline{x} \mapsto x \\ 0 \mapsto 1 \end{bmatrix} \quad \overline{t_0}\colon \overline{x} \mapsto \overline{x}; \begin{bmatrix} 1 \mapsto 0 \end{bmatrix} \quad t_0\colon x \mapsto 0; \begin{bmatrix} x \mapsto \bot \\ \overline{x} \mapsto 0 \\ 1 \mapsto \bot \end{bmatrix} \quad t_1\colon x \mapsto 1; \begin{bmatrix} x \mapsto \bot \\ \overline{x} \mapsto \bot \\ 0 \mapsto \bot \end{bmatrix}.$$

Intuitively, $\mathcal{P}$ repeatedly halves the number of agents in state x, and it accepts iff it never obtains an odd remainder. More precisely, the transitions of $\mathcal{P}$ are intended to be fired as follows, where C denotes the current configuration:

```
while $C(x) \neq 1$:
  while $C(x) \geq 2$: fire $s$      /*  split agents equally from $x$ to $\overline{x}$ and $0$ */
  if $C(x) = 0$: fire $\overline{s}$       /* move agents from $\overline{x}$ to $x$ if no remainder */
if $C(\overline{x}) = 0$: fire $t_1$         /*            if no remainder, then accept */
else: fire $\overline{t_0}\,t_0$             /*                        otherwise, reject */
```

It is easy to show that $\mathcal{P}$ produces a (lasting) consensus, and the right one, if transitions are executed as above. However, an arbitrary execution may not follow the above procedure. Firing transition $\overline{t_0}$ when not intended has no incidence on the outcome. Moreover, if another transition is fired when it should not be, then $\overline{s}$, t_0 or t_1 will detect this error by moving an agent to state $\bot$. In this case, by fairness, r eventually resets the agents back to the initial configuration and, again by fairness, transitions are eventually fired as intended.

▶ **Proposition 3.** *The broadcast consensus protocol* $\mathcal{P}$ *described above computes the predicate* φ*, defined as* $\varphi(x) = 1$ *iff* $x > 1$ *and* x *is a power of two.*

Leaderless broadcast protocols. A broadcast protocol $\mathcal{P} = (Q, R, B, \Sigma, L, I, O)$ is *leaderless* if $L = \mathbf{0}$. It can be shown that leaderless broadcast consensus protocols compute the same predicates as the general class. We only sketch the argument. First, a broadcast protocol with leader multiset L can be simulated by a protocol with a single leader. Indeed, the protocol can be designed so that the first task of the leader is to "recruit" the other leaders of L from among the agents. Second, a protocol with one leader can be simulated by a leaderless protocol because, loosely speaking, a broadcast protocol can elect a leader in a single computation step[2]. Indeed, if initially all agents are in a state, say q, then a broadcast $q \mapsto \ell; f$, where $f(q) = q'$, sends exactly one agent to leader state ℓ, and all other agents to state q'. It is simple to construct $\mathcal{P}'$ using this feature, and the details are omitted.

In the rest of the paper, we use protocols with leaders to simplify the constructions, but all results (except Proposition 17) remain valid for leaderless protocols.

4 Broadcast consensus protocols compute exactly NL

In this section, we prove our main theorem: a predicate is computable by a broadcast consensus protocol iff it is in NL. We follow the convention and say that a predicate φ belongs to NL if there is a nondeterministic Turing machine that accepts in $\mathcal{O}(\log n)$-space exactly the tuples $(x_1, x_2, \ldots, x_k) \in \mathbb{N}^k$, encoded in unary, such that $\varphi(x_1, x_2, \ldots, x_k)$ holds.

The proof is divided in two parts. Section 4.1 proves the easier direction: predicates computable by broadcast consensus protocols are in NL. Section 4.2 proves the converse, which is more involved.

4.1 Predicates computable by broadcast consensus protocols are in NL

We prove the result in more generality. We define a generic computational model in which the possible steps between configurations are given by an arbitrary relation preserving the number of agents. Formally, a *generic consensus protocol* is a tuple $\mathcal{P} = (Q, \text{Step}, \Sigma, L, I, O)$ where Q, Σ, L, I, O are defined as for population protocols, and $\text{Step} \subseteq \text{Pop}(Q) \times \text{Pop}(Q)$ is the *step relation* between populations, satisfying $|C| = |C'|$ for every $(C, C') \in \text{Step}$.

Clearly, broadcast consensus protocols are generic consensus protocols. Further, it is easy to see that if Step is the one-step relation of a broadcast protocol, then $\text{Step} \in \mathsf{NL}$. Indeed, $\text{Step} \in \mathsf{NL}$ if there is a nondeterministic Turing machine that given a pair of configurations (C, C') with n agents, uses $\mathcal{O}(\log n)$ space and accepts iff $(C, C') \in \text{Step}$. A quick inspection of the two conditions in the definition of Step (Definition 2) shows that this is the case.

Thus, it suffices to prove that generic consensus protocols satisfying $\text{Step} \in \mathsf{NL}$ can only compute predicates in NL. We sketch the proof, more details can be found in the full version of the paper.

▶ **Proposition 4.** *Let $\mathcal{P} = (Q, \text{Step}, \Sigma, L, I, O)$ be a generic consensus protocol computing a predicate φ. If $\text{Step} \in \mathsf{NL}$, then $\varphi \in \mathsf{NL}$. In particular, predicates computable by broadcast consensus protocols are in NL.*

Proof. We show that there is a nondeterministic Turing machine that decides whether $\varphi(\boldsymbol{x}) = 1$ holds, and uses $\mathcal{O}(\log |\boldsymbol{x}|)$ space. Let $G = (V, E)$ be the graph where V is the set of all configurations of $\mathcal{P}$ of size $|\boldsymbol{x}|$, and $(C, C') \in E$ iff $C \to C'$.

[2] Unlike population protocols, where efficient leader election is non-trivial and much studied; see e.g. [14].

It is easy to see that $\varphi(\boldsymbol{x}) = 1$ iff G contains a configuration C of size $|C| = |I(\boldsymbol{x})| = |\boldsymbol{x}|$ satisfying (1) $C_0 \xrightarrow{*} C$; and (2) every configuration reachable from C, including C itself, is a 1-consensus. Therefore, we can decide $\varphi(\boldsymbol{x}) = 1$ by guessing C, and checking (1) and (2) in $\mathcal{O}(\log |I(\boldsymbol{x})|)$ space. For (1), this follows from the fact that graph reachability is in NL. For (2), we observe that determining whether some configuration reachable from C is not a 1-consensus can be done in NL, and we use the fact that NL = coNL [20]. ◀

▶ Remark 5. Protocols with absence detector [22] are a class of generic consensus protocols, and hence Proposition 4 can be used to give an alternative proof of the fact that these protocols only compute predicates in NL.

4.2 Predicates in NL are computable by broadcast consensus protocols

The proof is involved, and we start by describing its structure. In Section 4.2.1, we show that it suffices to prove that every predicate in NL is *silently semi-computable*. In the rest of the section, we proceed to prove this in three steps. Loosely speaking, we show that:

- predicates computable by nondeterministic Turing machines in $\mathcal{O}(n)$ space can also be computed by counter machines with counters polynomially bounded in n (Section 4.2.2);
- predicates computed by polynomially bounded counter machines can also be computed by n-bounded counter machines, i.e. in which the sum of the values of all counters never exceeds their initial sum (Section 4.2.3);
- predicates computed by n-bounded counter machines can be silently semi-computed by broadcast protocols. (Section 4.2.4).

Finally, Section 4.2.5 puts all parts of the proof together.

4.2.1 Silent semi-computation

Recall that, loosely speaking, a protocol computes φ if it converges to 1 for inputs that satisfy φ, and it converges to 0 for inputs that do not satisfy φ. Additionally, a protocol *silently computes* φ if convergence to $b \in \{0, 1\}$ happens by reaching a *terminal b-consensus*, i.e., a configuration C that is a b-consensus and from which one can only reach C itself. (Intuitively, the protocol eventually becomes "silent" because no agent changes state anymore, and hence communication "stops".) We say that a protocol *silently semi-computes* φ if it reaches a terminal 1-consensus for inputs that satisfy φ, and no terminal configuration for other inputs.

▶ **Definition 6.** *A broadcast consensus protocol $\mathcal{P}$* silently semi-computes *a k-ary predicate φ if for every $\boldsymbol{\alpha} \in \mathbb{N}^k$ the following properties hold:*

1. *if $\varphi(\boldsymbol{\alpha}) = 1$, then every fair execution of $\mathcal{P}$ starting at $I(\boldsymbol{\alpha})$ eventually reaches a terminal 1-consensus configuration;*
2. *if $\varphi(\boldsymbol{\alpha}) = 0$, then no fair execution of $\mathcal{P}$ starting at $I(\boldsymbol{\alpha})$ eventually reaches a terminal configuration.*[3]

We show that if a predicate and its complement are both silently semi-computable by broadcast consensus protocols, say $\mathcal{P}_1$ and $\mathcal{P}_0$, then the predicate is also computable by a broadcast consensus protocol $\mathcal{P}$ which, intuitively, behaves as follows under input $\boldsymbol{\alpha}$. At every moment in time, $\mathcal{P}$ is simulating either $\mathcal{P}_1$ or $\mathcal{P}_0$. Initially, $\mathcal{P}$ simulates $\mathcal{P}_0$. Assume $\mathcal{P}$ is simulating $\mathcal{P}_i$ and the current configuration is C. If C is a terminal configuration of

[3] Since every finite execution can be extended to a fair one, this condition is actually equivalent to "no terminal configuration is reachable from $I(\boldsymbol{\alpha})$".

$\mathcal{P}_i$, then $\mathcal{P}$ terminates too. Otherwise, $\mathcal{P}$ nondeterministically chooses one of three options: continue thesimulation of $\mathcal{P}_i$, "reset" the computation to $I_0(\boldsymbol{\alpha})$, i.e., start simulating $\mathcal{P}_0$, or "reset" the computation to $I_1(\boldsymbol{\alpha})$. Conditions 1 and 2 ensure that exactly one of $\mathcal{P}_0$ and $\mathcal{P}_1$ can reach a terminal configuration, namely $\mathcal{P}_{\varphi(\boldsymbol{\alpha})}$. Fairness ensures that $\mathcal{P}$ will eventually reach a terminal configuration of $\mathcal{P}_{\varphi(\boldsymbol{\alpha})}$, and so, by condition 1, that it will always reach the right consensus. Hence, $\mathcal{P}$ silently computes φ.

The "reset" is implemented by means of a broadcast that sends every agent to its initial state in the configuration $I_j(\boldsymbol{\alpha})$; for this, the states of $\mathcal{P}$ are partitioned into classes, one for each input symbol $x \in X$. Every agent moves only within the states of one of the classes, and so every agent "remembers" its initial state in both $\mathcal{P}_0$ and $\mathcal{P}_1$.

▶ **Lemma 7.** *Let φ be an m-ary predicate, and let $\overline{\varphi}$ be the predicate defined by $\overline{\varphi}(\boldsymbol{\alpha}) \overset{def}{=} 1 - \varphi(\boldsymbol{\alpha})$ for every $\boldsymbol{\alpha} \in \mathbb{N}^m$. Further let $\mathcal{P}_1$ and $\mathcal{P}_0$ be broadcast consensus protocols that silently semi-compute φ and $\overline{\varphi}$, respectively. The following holds: there exists a broadcast consensus protocol $\mathcal{P}$ that silently computes φ.*

Proof. Let $\mathcal{P}_1 = (Q_1, R_1, B_1, \Sigma, I_1, O_1)$ and $\mathcal{P}_0 = (Q_0, R_0, B_0, \Sigma, I_0, O_0)$ be protocols that silently semi-compute φ and $\overline{\varphi}$, respectively. Assume w.l.o.g. that Q_1 and Q_0 are disjoint. We construct a protocol $\mathcal{P} = (Q, R, B, \Sigma, I, O)$ that computes φ.

For the sake of clarity we refrain from giving a fully formal description, but we provide enough details to show that the design idea above can indeed be implemented.

States and mappings. The set of states of $\mathcal{P}$ is defined as:

$$Q \overset{\text{def}}{=} \Sigma \times (Q_1 \cup Q_0 \cup \{\texttt{reset}\})$$

If an agent is in state (x, q), we say that x is its *origin* and that q is its *position.* The initial position of an agent is its initial state in $\mathcal{P}_0$, i.e. $I(x) \overset{\text{def}}{=} (x, I_0(x))$. Transitions will be designed so that agents may update their position, but not their origin. Alternatively, instead of applying a transition, agents can nondeterministically choose to transition from $(x, q) \in X \times (Q_1 \cup Q_0)$ to $(x, \texttt{reset})$. An agent in state $(x, \texttt{reset})$ eventually resets the simulation to either $\mathcal{P}_0$ or $\mathcal{P}_1$.

Simulation transitions. We define transitions that proceed with the simulation of $\mathcal{P}_0$ and $\mathcal{P}_1$ as follows. For every $i \in \{1, 0\}$, every $x, y \in \Sigma$, and every non-silent rendez-vous transition $(q, r) \mapsto (q', r')$ of R_i, we add the following rendez-vous transitions to R:

$$(x, q), (y, r) \mapsto (x, q'), (y, r') \qquad \text{and} \qquad (x, q), (y, r) \mapsto (x, \texttt{reset}), (y, \texttt{reset}).$$

The first transition implements the simulation, while the second transition enables resets when the simulation has not reached a terminal configuration. For every broadcast transition $q \mapsto q'; f$ of B_i and every $x \in \Sigma$, we add the following broadcast transitions to B:

$$(x, q) \mapsto (x, q'); f'$$
$$(x, q) \mapsto (x, \texttt{reset}); f'$$

where f' only acts on Q_i by $f'(y, r) \overset{\text{def}}{=} (y, f(r))$ for every $(y, r) \in \Sigma \times Q_i$. The first transition implements the simulation of a broadcast in the original protocols, while the second transition enables a reset.

Reset transitions. We define transitions that trigger a new simulation of either $\mathcal{P}_0$ or $\mathcal{P}_1$. For every $i \in \{1, 0\}$, let $f_i : Q \to Q$ be the function defined as $f_i(x, q) \overset{\text{def}}{=} (x, I_i(x))$ for every $(x, q) \in Q$. For every $i \in \{1, 0\}$ and every $x \in \Sigma$, we add the following broadcast transition to B: $(x, \texttt{reset}) \mapsto (x, I_i(x)); f_i$. ◀

Using Lemma 7, we may now prove the following:

▶ **Proposition 8.** *If every predicate in* NL *is silently semi-computable by broadcast consensus protocols, then every predicate in* NL *is silently computable (and so computable) by broadcast consensus protocols.*

Proof. Assume every predicate in NL is silently semi-computable by broadcast consensus protocols, and let φ be a predicate in NL. We resort to the powerful result stating that predicates in coNL and NL coincide. This is an immediate corollary of the coNL = NL theorem for languages [20, 26, 23], and the fact that one can check in constant space whether a given word encodes a vector of natural numbers of fixed arity. Thus, both φ and $\overline{\varphi}$ are predicates in NL, and so, by assumption, silently semi-computable by broadcast consensus protocols. By Lemma 7, they are silently computable by broadcast consensus protocols. ◀

4.2.2 Simulation of Turing machines by counter machines

We recall that nondeterministic Turing machines working in $\mathcal{O}(n)$ space can be simulated by counter machines whose counters are polynomially bounded in n, and so that both models compute the same predicates.

Let $X = \{x_1, x_2, \ldots, x_k\}$ and $\mathit{Ins} = \{\texttt{inc}(x), \texttt{dec}(x), \texttt{zro}(x), \texttt{nzr}(x), \texttt{nop} \mid x \in X\}$. A *$k$-counter machine* $\mathcal{M}$ over *counters* X is a tuple $(Q, X, \Delta, m, q_0, q_a, q_r)$, where Q is a finite set of *control states*; $\Delta \subseteq Q \times \mathit{Ins} \times Q$ is the *transition relation*; $m \leq k$ is the *number of input counters*; and q_0, q_a, q_r are the *initial*, *accepting*, and *rejecting states*, respectively.

A configuration of $\mathcal{M}$ is a pair $C = (q, \boldsymbol{v}) \in Q \times \mathbb{N}^k$ consisting of a control state q and counter values $\boldsymbol{v}$. For every $i \in [k]$, we denote the value of counter x_i in C by $C(x_i) \overset{\text{def}}{=} \boldsymbol{v}_i$. The *size* of C is $|C| \overset{\text{def}}{=} \sum_{i=1}^{k} C(x_i)$.

Let $\boldsymbol{e}_i$ be the i-th row of the $k \times k$ identity matrix. Given $\mathit{ins} \in \mathit{Ins}$, we define the relation $\xrightarrow{\mathit{ins}}$ over configurations as follows: $(q, \boldsymbol{v}) \xrightarrow{\mathit{ins}} (q', \boldsymbol{v}')$ iff $(q, \mathit{ins}, q') \in \Delta$ and one of the following holds: $\mathit{ins} = \texttt{inc}(x_i)$ and $\boldsymbol{v}' = \boldsymbol{v} + \boldsymbol{e}_i$; $\mathit{ins} = \texttt{dec}(x_i)$, $\boldsymbol{v}_i > 0$, and $\boldsymbol{v}' = \boldsymbol{v} - \boldsymbol{e}_i$; $\mathit{ins} = \texttt{zro}(x_i)$, $\boldsymbol{v}_i = 0$, and $\boldsymbol{v}' = \boldsymbol{v}$; $\mathit{ins} = \texttt{nzr}(x_i)$, $\boldsymbol{v}_i > 0$, and $\boldsymbol{v}' = \boldsymbol{v}$; $\mathit{ins} = \texttt{nop}$ and $\boldsymbol{v}' = \boldsymbol{v}$.

For every $\boldsymbol{\alpha} \in \mathbb{N}^m$, the initial configuration of $\mathcal{M}$ with input $\boldsymbol{\alpha}$ is defined as:

$$C_{\boldsymbol{\alpha}} \overset{\text{def}}{=} (q_0, (\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, \ldots, \boldsymbol{\alpha}_m, \underbrace{0, \ldots, 0}_{k-m \text{ times}})).$$

We say $\mathcal{M}$ *accepts* $\boldsymbol{\alpha}$ if there exist counter values $\boldsymbol{v} \in \mathbb{N}^k$ satisfying $C_{\boldsymbol{\alpha}} \xrightarrow{*} (q_a, \boldsymbol{v})$. We say $\mathcal{M}$ *rejects* $\boldsymbol{\alpha}$ if M does not accept $\boldsymbol{\alpha}$ and for all configurations C' with $C_{\boldsymbol{\alpha}} \xrightarrow{*} C'$, there exists $\boldsymbol{v} \in \mathbb{N}^k$ satisfying $C' \xrightarrow{*} (q_r, \boldsymbol{v})$. We say $\mathcal{M}$ *computes* a predicate $\varphi \colon \mathbb{N}^m \to \{0, 1\}$ if $\mathcal{M}$ accepts all inputs $\boldsymbol{\alpha}$ such that $\varphi(\boldsymbol{\alpha}) = 1$, and rejects all $\boldsymbol{\alpha}$ such that $\varphi(\boldsymbol{\alpha}) = 0$.

A counter machine $\mathcal{M}$ is $f(n)$*-bounded* if $|C| \leq f(|C_{\boldsymbol{\alpha}}|)$ holds for every initial configuration $C_{\boldsymbol{\alpha}}$ and every configuration C reachable from $C_{\boldsymbol{\alpha}}$. It is well-known that counter machines can simulate Turing machines:

▶ **Theorem 9** ([19, Theorem 3.1])**.** *A predicate is computable by an $s(n)$-space-bounded Turing machine iff it is computable by a $2^{s(n)}$-bounded counter machine.*

In [19], a weaker version of Theorem 9 is proven that applies to deterministic Turing and counter machines only. However, the proof can be easily adapted to the nondeterministic setting we consider here.

▶ **Corollary 10.** *A predicate is in NL iff it is computable by a polynomially bounded counter machine.*

4.2.3 Simulation of polynomially bounded counter machines by n-bounded counter machines

▶ **Lemma 11.** *For every polynomially bounded counter machine that computes some predicate φ, there exists an n-bounded counter machine that computes φ.*

Proof. We sketch the main idea of the proof; details can be found in the full version of the paper. Let $c \in \mathbb{N}_{>0}$ and let $\mathcal{M}$ be an n^c-bounded counter machine with k counters. To simulate $\mathcal{M}$ by an n-bounded counter machine $\overline{\mathcal{M}}$, we need some way to represent any value $\ell \in [0, n^c]$ by means of counters with values in $[0, n]$. We encode such a value ℓ by its base $n+1$ representation over c counters. Zero-tests are performed by zero-testing all c counters sequentially. Nonzero-tests are implemented similarly with parallel tests. Incrementation and decrementation are implemented with gadgets to (a) assign 0 to a counter; (b) assign n to a counter; (c) test whether a counter value equals n.

This construction is only *weakly* n-bounded, in the sense that all counters are indeed bounded by n, but the overall sum can reach $k \cdot n$. To circumvent this issue, we simulate $\overline{\mathcal{M}}$ by another counter machine $\mathcal{M}'$ whose counters symbolically hold values from multiple counters of $\overline{\mathcal{M}}$. In more details, the counters are defined as $\{y_S : S \subseteq \overline{X}\}$. Intuitively, if counter y_S has value a, then it contributes by a to the value of each counter of S. For example, if $\overline{X} = \{x_1, x_2, x_3\}$ and the input size is $n = 6$, then counter values $(x_1, x_2, x_3) = (6, 1, 4)$ of $\overline{\mathcal{M}}$ can be represented in $\mathcal{M}'$ as $y_{\{x_1,x_2,x_3\}} = 1$, $y_{\{x_1,x_3\}} = 3$, $y_{\{x_1\}} = 2$, and $y_S = 0$ for every other S. Under such a representation, the sum of all counters equals n. Moreover, all instructions can be implemented quite easily. ◀

4.2.4 Simulation of n-bounded counter machines by broadcast consensus protocols

Let $\mathcal{M} = (Q, X, \Delta, m, q_0, q_a, q_r)$ be an n-bounded counter machine that computes some predicate $\varphi \colon \mathbb{N}^m \to \{0, 1\}$. We construct a broadcast protocol $\mathcal{P} = (Q', R, B, \Sigma, L, I, O)$ that silently semi-computes φ.

States and mappings. Let $X' \overset{\text{def}}{=} X \cup \{\mathit{idle}, \mathit{err}\}$. The states of $\mathcal{P}$ are defined as

$$Q' \overset{\text{def}}{=} \underbrace{Q \times \{0,1\}}_{\text{leader states}} \cup \underbrace{X' \times X \times \{0,1\}}_{\text{nonleader states}}.$$

The protocol will be designed in such a way that there is always exactly one agent, called the *leader*, in states $Q \times \{0, 1\}$. Whenever the leader is in state (q, b), we say that its *position* is q, and its *opinion* is b. Every other agent will remain in a state from $X' \times X \times \{0, 1\}$. Whenever a nonleader agent is in state (x, y, b), we say that its *position* is x, its *origin* is y, and its *opinion* is b. Intuitively, the leader is in charge of storing the control state of $\mathcal{M}$, and the nonleaders are in charge of storing the counter values of $\mathcal{M}$.

The protocol has a single leader whose initial position is the initial control state of $\mathcal{M}$, i.e. $L \overset{\text{def}}{=} \lbag (q_0, 0) \rbag$. Moreover, every nonleader agent initially has its origin set to its initial position, which will remain unchanged by definition of the forthcoming transition relation: $I(x) \overset{\text{def}}{=} (x, x, 0)$ for every $x \in X$. The output of each agent is its opinion:

$$\begin{aligned} O(q, b) &\overset{\text{def}}{=} b \\ O(x, y, b) &\overset{\text{def}}{=} b \end{aligned} \quad \text{for every } q \in Q, x \in X', y \in X, b \in \{0, 1\}.$$

We now describe how $\mathcal{P}$ simulates the instructions of $\mathcal{M}$.

Decrementation/incrementation. For every transition $q \xrightarrow{\texttt{dec}(x)} r \in \Delta$, every $y \in X$ and every $b, b' \in \{0, 1\}$, we add to R the rendez-vous transition:

$$(q, b), (x, y, b') \mapsto (r, b), (\mathit{idle}, y, b').$$

These transitions change the position of one agent from x to *idle*, and thus decrement the number of agents in position x.

Similarly, for every transition $q \xrightarrow{\texttt{inc}(x)} r$, every $y \in X$ and every $b, b' \in \{0, 1\}$, we add to R the rendez-vous transition:

$$(q, b), (\mathit{idle}, y, b') \mapsto (r, b), (x, y, b').$$

These transitions change the position of an idle agent to x, and thus increment the number of agents in position x. If no agent is in position *err*, then at least one idle agent is available when a counter needs to be incremented, since $\mathcal{M}$ is n-bounded.

Nonzero-tests. For every $q \xrightarrow{\texttt{nzr}(x)} r \in \Delta$, every $y \in X$ and every $b, b' \in \{0, 1\}$, we add to R the rendez-vous transition:

$$(q, b), (x, y, b') \mapsto (r, b), (x, y, b').$$

These transitions can only be executed if there is at least one agent in position x, and thus only if the value of x is nonzero.

Zero-tests. For a given $x \in X$, let $f^x_{err} \colon Q' \to Q'$ be the function that maps every nonleader in position x to the error position, i.e. $f^x_{err}(x, y, b) \overset{\text{def}}{=} (\mathit{err}, y, b)$ for every $y \in X, b \in \{0, 1\}$, and f^x is the identity for all other states.

For every transition $q \xrightarrow{\texttt{zro}(x)} r \in \Delta$ and every $b \in \{0, 1\}$, we add to B the broadcast transition $(q, b) \mapsto (r, b); f^x_{err}$. If such a transition occurs, then nonleaders in position x move to *err*. Thus, an error is detected iff the value of x is nonzero.

To recover from errors, $\mathcal{P}$ can be reset to its initial configuration as follows. Let $f_{rst} \colon Q' \to Q'$ be the function that sends every state back to its origin, i.e.

$$\begin{aligned} f_{rst}(q, b) &\overset{\text{def}}{=} (q_0, 0) && \text{for every } q \in Q, b \in \{0, 1\}, \\ f_{rst}(x, y, b) &\overset{\text{def}}{=} (y, y, 0) && \text{for every } x \in X', y \in X, b \in \{0, 1\}. \end{aligned}$$

For every $y \in X$ and every $b \in \{0, 1\}$, we add the following broadcast transition to B to reset $\mathcal{P}$ to its initial configuration:

$$(\mathit{err}, y, b) \mapsto (y, y, 0); f_{rst}.$$

Acceptance. For every $q \in Q \setminus \{q_a\}$ and $b \in \{0,1\}$, we add to B the broadcast transition $(q,b) \mapsto (q_0,0); f_{rst}$. Intuitively, as long as the leader's position differs from the accepting control state q_a, it can reset $\mathcal{P}$ to its initial configuration. This ensures that $\mathcal{P}$ can try *all* computations.

Let $f_{err} \colon Q' \to Q'$ be the function that changes the opinion of each state to 1, i.e.

$$\begin{aligned} f_{err}(q,b) &\stackrel{\text{def}}{=} (q,1) && \text{for every } q \in Q, b \in \{0,1\}, \\ f_{err}(x,y,b) &\stackrel{\text{def}}{=} (x,y,1) && \text{for every } x \in X', y \in X, b \in \{0,1\}. \end{aligned}$$

For every $b \in \{0,1\}$, we add the following transition to B:

$$t_{one,b} \colon (q_a,b) \to (q_a,1); f_{one}.$$

Intuitively, these transitions change the opinion of every agent to 1. If such a transition occurs in a configuration with no agent in *err*, then no agent can change its state anymore, and the stable consensus 1 has been reached.

Correctness. Let us fix some some input $\boldsymbol{\alpha} \in \mathbb{N}^m$. Let C_0 and D_0 be respectively the initial configurations of $\mathcal{M}$ and $\mathcal{P}$ on input $\boldsymbol{\alpha}$. Abusing notation, for every $D \in \mathrm{Pop}(Q')$, let

$$D(x) \stackrel{\text{def}}{=} \sum_{(x,y,b) \in Q'} D(x,y,b).$$

The two following propositions state that every execution of $\mathcal{M}$ has a corresponding execution in $\mathcal{P}$ and vice versa. The proofs are routine.

▶ **Proposition 12.** *Let C be a configuration of $\mathcal{M}$ such that C is in control state q and $C_0 \xrightarrow{*} C$. There exists a configuration $D \in \mathrm{Pop}(Q')$ such that (i) $D_0 \xrightarrow{*} D$; (ii) $D(x) = C(x)$ for every $x \in X$; (iii) $D(err) = 0$; and (iv) $D(q,b) = 1$ for some $b \in \{0,1\}$.*

▶ **Proposition 13.** *Let $D \in \mathrm{Pop}(Q')$ be such that $D_0 \xrightarrow{*} D$. If $D(err) = 0$, then there is a configuration C of $\mathcal{M}$ such that (i) $C_0 \xrightarrow{*} C$; (ii) $C(x) = D(x)$ for every $x \in X$; and (iii) if $D(q,b) = 1$ for some $(q,b) \in Q'$, then C is in control state q.*

We may now prove that $\mathcal{P}$ silently semi-computes φ.

▶ **Proposition 14.** *For every n-bounded counter machine $\mathcal{M}$ that computes some predicate φ, there exists a broadcast consensus protocol that silently semi-computes φ.*

Proof. We show that $\mathcal{P}$ silently semi-computes φ by proving the two properties of Definition 6. Let $\boldsymbol{\alpha}$ be an input.

1. Assume $\varphi(\boldsymbol{\alpha}) = 1$. Then $\mathcal{M}$ accepts $\boldsymbol{\alpha}$, and so there is a configuration C such that $C_0 \xrightarrow{*} C$ and C is in control state q_a. By Proposition 12, there exists some configuration $D \in \mathrm{Pop}(Q')$ satisfying $D_0 \xrightarrow{*} D$, $D(err) = 0$ and $D(q_a, b) = 1$. Since $\mathcal{M}$ halts when reaching q_a, the only transition enabled at D is $t_{one,b}$, and its application yields a terminal configuration D' of consensus 1. Further, every configuration reachable from D_0, where the leader is not in position q_a or where some nonleader is in position *err*, can be set back to D_0 via some reset transition. Therefore, every fair execution of $\mathcal{P}$ starting at $I(\boldsymbol{\alpha}) = C_0$ will eventually reach D'.
2. Assume $\varphi(\boldsymbol{\alpha}) = 0$. We prove by contradiction that no configuration D reachable from D_0 is terminal. Assume the contrary. We must have $D(q_a, 1) = 1$, $D(err) = 0$ and $O(D) = 1$, for otherwise some broadcast transition with f_{rst} or f_{one} would be enabled. From this and by Proposition 13, there exists some configuration C of $\mathcal{M}$ in control state q_a and satisfying $C_0 \xrightarrow{*} C$. Thus, $\mathcal{M}$ accepts $\boldsymbol{\alpha}$, contradicting $\varphi(\alpha) = 0$. ◀

4.2.5 Main theorem

We prove our main result, namely that broadcast consensus protocols precisely compute the predicates in NL.

▶ **Theorem 15.** *Broadcast consensus protocols compute exactly the predicates in* NL*.*

Proof. Proposition 4 shows that every predicate computable by broadcast consensus protocols is in NL. For the other direction, let φ be a predicate in NL. Since NL = coNL by Immerman-Stelepcsényi's theorem, the complement predicate $\overline{\varphi}$ is also in NL. Thus, φ and $\overline{\varphi}$ are computable by $\mathcal{O}(\log n)$-space-bounded nondeterministic Turing machines. By Theorem 9 and Proposition 11, φ and $\overline{\varphi}$ are computable by polynomially bounded counter machines, and thus by n-bounded counter machines. Therefore, by Proposition 14, φ and $\overline{\varphi}$ are silently semi-computable by broadcast consensus protocols. By Proposition 8, this implies that φ is silently computable by a broadcast consensus protocol. ◀

Actually, the proof shows this slightly stronger result:

▶ **Corollary 16.** *A predicate is computable by a broadcast consensus protocol iff it is silently computable by a broadcast consensus protocol. In particular, broadcast consensus protocols silently compute all predicates in* NL*.*

5 Subclasses of broadcast consensus protocols

While broadcasting is a natural, well understood, and much used communication mechanism, it also consumes far more energy than rendez-vous communication. In particular, agents able to broadcast are more expensive to implement. In this section, we briefly analyze which restrictions can be imposed on the broadcast model without reducing its computational power. We show that all predicates in NL can be computed by protocols satisfying two properties:

1. only one agent broadcasts; all other agents only use rendez-vous communication.
2. the broadcasting agent only needs to send one signal, meaning that the receivers' response is independent of the broadcast signal.

Finally, we show that a third restriction *does* decrease the computational power. In simulations of the previous section, broadcasts are often used to "reset" the system. Since computational models with resets have been devoted quite some attention [21, 25, 13, 11], we investigate the computational power of protocols with resets.

Protocols with only one broadcasting agent. Loosely speaking, a broadcast protocol with one broadcasting agent is a broadcast protocol $\mathcal{P} = (Q, R, B, \Sigma, L, I, O)$ with a set Q_ℓ of leader states such that $L = \wr q \wr$ for some $q \in Q_\ell$ (i.e., there is exactly one leader), and whose transitions ensure that the leader always remains within Q_ℓ, that no other agent enters Q_ℓ, and that only agents in Q_ℓ can trigger broadcast transitions. Protocols with multiple broadcasting agents can be simulated by protocols with one broadcasting agent, say b. Instead of directly broadcasting, an agent communicates with b by rendez-vous, and delegates to b the task of executing the broadcast. More precisely, a broadcast transition $q \mapsto q'; f$ is simulated by a rendez-vous transition $(q, q_\ell) \mapsto (q_{aux}, q_{\ell,f})$, followed by a broadcast transition $q_{\ell,f} \mapsto q_\ell; (f \cup \{q_{aux} \mapsto q'\})$.

Single-signal broadcast protocols. In single-signal protocols the receivers' response is independent of the broadcast signal. Formally, a broadcast protocol (Q, R, B, Σ, I, O) is a *single-signal protocol* if there exists a function $f : Q \to Q$ such that $B \subseteq Q^2 \times \{f\}$.

▶ **Proposition 17.** *Predicates computable by broadcast consensus protocols are also computable by single-signal broadcast protocols.*

Proof. We give a proof sketch; details can be found in the full version of the paper. We simulate a broadcast protocol $\mathcal{P}$ by a single-signal protocol $\mathcal{P}'$. The main point is to simulate a broadcast step $C_1 \xrightarrow{q_1 \mapsto q_2; g} C_2$ of $\mathcal{P}$ by a sequence of steps of $\mathcal{P}'$.

In $\mathcal{P}$, an agent at state q_1, say a, moves to q_2, and broadcasts the signal with meaning "react according to g". Intuitively, in $\mathcal{P}'$, agent a broadcasts the unique signal of $\mathcal{P}'$, which has the meaning "freeze". An agent that receives the signal, say b, becomes "frozen". Frozen agents can only be "awoken" by a rendez-vous with a. When the rendez-vous happens, a tells b which state it has to move to according to g.

The problem with this procedure is that a has no way to know if it has already performed a rendez-vous with all frozen agents. Thus, frozen agents can spontaneously move to a state *err* indicating "I am tired of waiting". If an agent is in this state, then eventually all agents go back to their initial states, reinitializing the computation. This is achieved by letting agents in state *err* move to their initial states while broadcasting the "freeze" signal. ◀

Protocols with reset. In protocols with reset, all broadcasts transitions reset the protocol to its initial configuration. Formally, a *population protocol with reset* is a broadcast protocol $\mathcal{P} = (Q, R, B, \Sigma, I, O)$ such that for every finite execution $C_0 C_1 \cdots C_k$ from an initial configuration C_0, the following holds: $C_k \xrightarrow{b} C'$ implies $C' = C_0$ for every $b \in B$ and every $C' \in \mathrm{Pop}(Q)$.

▶ **Proposition 18.** *Every predicate computable by a population protocol with reset is Presburger-definable, and thus computable by a standard population protocol.*

Proof. We give a proof sketch; details can be found in the full version of the paper. Let $\mathcal{P} = (Q, R, B, \Sigma, I, O)$ be a population protocol with reset that computes some predicate. We show that the set of accepting initial configurations of $\mathcal{P}$, denoted I_1, is Presburger-definable as follows. Let:

- $\mathcal{P}'$ be the population protocol obtained from $\mathcal{P}$ by eliminating the resets;
- $\mathcal{N}$ be the set of configurations C of $\mathcal{P}'$ from which no reset can occur, i.e., no configuration reachable from C enables a reset of $\mathcal{P}$;
- S_1 be the set of configurations C of $\mathcal{P}'$ that are stable 1-consensuses, i.e., $O(C') = 1$ for every C' reachable from C;
- $\mathcal{B}$ be the set of configurations C of $\mathcal{P}'$ that belong to a bottom strongly connected component of the configuration graph, i.e., C can reach C' iff C' can reach C.

We show that an initial configuration C belongs to I_1 iff it belongs to S_1 or it can reach a configuration from $S_1 \cap \mathcal{B} \cap \mathcal{N}$. Using results from [17], showing in particular that $\mathcal{B}$ is Presburger-definable, we show that I_1 is Presburger-definable. ◀

6 Conclusion

We have studied the expressive power of broadcast consensus protocols: an extension of population protocols with reliable broadcasts, a standard communication primitive in concurrency and distributed computing. We have shown that, despite their simplicity, they precisely compute predicates from the complexity class NL, and are thus as expressive as several other proposals from the literature which require a primitive more difficult to implement: *receiving* messages from all agents, instead of sending messages to all agents.

As future work, we wish to study properties beyond expressiveness, such as state complexity and space vs. speed trade-offs. It would also be interesting to tackle the formal verification of broadcast consensus protocols. Although this is challenging as it goes beyond Presburger arithmetic and the decidability frontier, it has recently been shown that models with broadcasts admit more tractable approximations [10].

References

1 Mehran Abolhasan, Tadeusz A. Wysocki, and Eryk Dutkiewicz. A review of routing protocols for mobile ad hoc networks. *Ad Hoc Networks*, 2(1):1–22, 2004. doi:10.1016/S1570-8705(03)00043-X.

2 Dan Alistarh, James Aspnes, David Eisenstat, Rati Gelashvili, and Ronald L. Rivest. Time-Space Trade-offs in Population Protocols. In *Proc. Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2560–2579, 2017. doi:10.1137/1.9781611974782.169.

3 Dan Alistarh, James Aspnes, and Rati Gelashvili. Space-Optimal Majority in Population Protocols. In *Proc. Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2221–2239, 2018. doi:10.1137/1.9781611975031.144.

4 Dan Alistarh and Rati Gelashvili. Recent Algorithmic Advances in Population Protocols. *SIGACT News*, 49(3):63–73, 2018. doi:10.1145/3289137.3289150.

5 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. In *Proc. 23rd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 290–299, 2004. doi:10.1145/1011767.1011810.

6 Dana Angluin, James Aspnes, Zoë Diamadi, Michael J. Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed Computing*, 18(4):235–253, 2006.

7 Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007. doi:10.1007/s00446-007-0040-2.

8 James Aspnes. Clocked Population Protocols. In *Proc. ACM Symposium on Principles of Distributed Computing (PODC)*, pages 431–440, 2017.

9 Nathalie Bertrand, Miheer Dewaskar, Blaise Genest, and Hugo Gimbert. Controlling a Population. In *Proc. 28th International Conference on Concurrency Theory (CONCUR)*, volume 85, pages 12:1–12:16, 2017. doi:10.4230/LIPIcs.CONCUR.2017.12.

10 Michael Blondin, Christoph Haase, and Filip Mazowiecki. Affine Extensions of Integer Vector Addition Systems with States. In *Proc. 29th International Conference on Concurrency Theory (CONCUR)*, pages 14:1–14:17, 2018. doi:10.4230/LIPIcs.CONCUR.2018.14.

11 Dmitry Chistikov, Christoph Haase, and Simon Halfon. Context-free commutative grammars with integer counters and resets. *Theoretical Computer Science*, 735:147–161, 2018. doi:10.1016/j.tcs.2016.06.017.

12 Giorgio Delzanno, Jean-François Raskin, and Laurent Van Begin. Towards the Automated Verification of Multithreaded Java Programs. In *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 173–187, 2002. doi:10.1007/3-540-46002-0_13.

13 Catherine Dufourd, Alain Finkel, and Philippe Schnoebelen. Reset Nets Between Decidability and Undecidability. In *Proc. 25th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 103–115, 1998. doi:10.1007/BFb0055044.

14 Robert Elsässer and Tomasz Radzik. Recent Results in Population Protocols for Exact Majority and Leader Election. *Bulletin of the EATCS*, 126, 2018.

15 E. Allen Emerson and Kedar S. Namjoshi. On Model Checking for Non-Deterministic Infinite-State Systems. In *Proc. Thirteenth Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 70–80, 1998. doi:10.1109/LICS.1998.705644.

16 Javier Esparza, Alain Finkel, and Richard Mayr. On the Verification of Broadcast Protocols. In *Proc. 14th Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 352–359, 1999. doi:10.1109/LICS.1999.782630.

17 Javier Esparza, Pierre Ganty, Jérôme Leroux, and Rupak Majumdar. Verification of population protocols. *Acta Informatica*, 54(2):191–215, 2017. doi:10.1007/s00236-016-0272-3.

18 Alain Finkel and Jérôme Leroux. How to Compose Presburger-Accelerations: Applications to Broadcast Protocols. In *Proc. 22nd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 145–156, 2002. doi:10.1007/3-540-36206-1_14.

19 Patrick C. Fischer, Albert R. Meyer, and Arnold L. Rosenberg. Counter Machines and Counter Languages. *Mathematical Systems Theory*, 2(3):265–283, 1968.

20 Neil Immerman. Nondeterministic Space is Closed Under Complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988. doi:10.1137/0217058.

21 David Lee and Mihalis Yannakakis. Testing Finite-State Machines: State Identification and Verification. *IEEE Transactions on Computers*, 43(3):306–320, 1994. doi:10.1109/12.272431.

22 Othon Michail and Paul G. Spirakis. Terminating population protocols via some minimal global knowledge assumptions. *Journal of Parallel and Distributed Computing*, 81-82:1–10, 2015.

23 Christos H. Papadimitriou. *Computational complexity.* Academic Internet Publ., 2007.

24 Sylvain Schmitz and Philippe Schnoebelen. The Power of Well-Structured Systems. In *Proc. 24th International Conference on Concurrency Theory (CONCUR)*, pages 5–24, 2013. doi:10.1007/978-3-642-40184-8_2.

25 Philippe Schnoebelen. Revisiting Ackermann-Hardness for Lossy Counter Machines and Reset Petri Nets. In *Proc. 35th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 616–628, 2010. doi:10.1007/978-3-642-15155-2_54.

26 Róbert Szelepcsényi. The Method of Forced Enumeration for Nondeterministic Automata. *Acta Informatica*, 26(3):279–284, 1988. doi:10.1007/BF00299636.

27 Jannis Uhlendorf, Agnès Miermont, Thierry Delaveau, Gilles Charvin, François Fages, Samuel Bottani, Pascal Hersen, and Gregory Batt. In silico control of biomolecular processes. In *Computational Methods in Synthetic Biology*, pages 277–285. Marchisio, Mario Andrea, 2015. doi:10.1007/978-1-4939-1878-2_13.

28 Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 2008. doi:10.1016/j.comnet.2008.04.002.

Reconfiguration and Message Losses in Parameterized Broadcast Networks

Nathalie Bertrand
Univ. Rennes, Inria, CNRS, IRISA – Rennes, France

Patricia Bouyer
LSV, CNRS & ENS Paris-Saclay, Univ. Paris-Saclay – Cachan, France

Anirban Majumdar
Univ. Rennes, Inria, CNRS, IRISA – Rennes, France
LSV, CNRS & ENS Paris-Saclay, Univ. Paris-Saclay – Cachan, France

Abstract

Broadcast networks allow one to model networks of identical nodes communicating through message broadcasts. Their parameterized verification aims at proving a property holds for any number of nodes, under any communication topology, and on all possible executions. We focus on the coverability problem which dually asks whether there exists an execution that visits a configuration exhibiting some given state of the broadcast protocol. Coverability is known to be undecidable for static networks, *i.e.* when the number of nodes and communication topology is fixed along executions. In contrast, it is decidable in PTIME when the communication topology may change arbitrarily along executions, that is for reconfigurable networks. Surprisingly, no lower nor upper bounds on the minimal number of nodes, or the minimal length of covering execution in reconfigurable networks, appear in the literature.

In this paper we show tight bounds for cutoff and length, which happen to be linear and quadratic, respectively, in the number of states of the protocol. We also introduce an intermediary model with static communication topology and non-deterministic message losses upon sending. We show that the same tight bounds apply to lossy networks, although, reconfigurable executions may be linearly more succinct than lossy executions. Finally, we show NP-completeness for the natural optimisation problem associated with the cutoff.

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

Keywords and phrases model checking, parameterized verification, broadcast networks

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.32

Funding The second author was supported by ERC project EQualIS (308087).

1 Introduction

Parameterized verification. Systems formed of many identical agents arise in many concrete areas: distributed algorithms, populations, communication or cache-coherence protocols, chemical reactions etc. Models for such systems depend on the communication or interaction means between the agents. For example pairwise interactions are commonly used for populations of individuals, whereas selective broadcast communications are more relevant for communication protocols on ad-hoc networks. The capacity of the agents, and thus models that are used to represent their behaviour also vary.

Verifying such systems amounts to checking that a property holds independently of the number of agents. Typically, a consensus algorithm should be correct for any number of participants. We refer to these systems as parameterized systems, and the parameter is the number of agents. The verification of parameterized systems started in the late 80's

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 32; pp. 32:1–32:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and recently regained attention from the model-checking community [11, 8, 6, 1]. It can be seen as particular cases of infinite-state-system verification, and the fact that all agents are identical can sometimes lead to efficient algorithms [5].

Broadcast networks. This paper targets the application to protocols over ad-hoc networks, and we thus focus on the model of broadcast networks [3]. A broadcast network is composed of several nodes that execute the same broadcast protocol. The latter is a finite automaton, where transitions are labeled with message sendings or message receptions. Configuration in broadcast networks is then comprised of a set of agents, their current local states, together with a communication topology (which represents which agents are within radio range). A transition represents the effect of one agent sending a message to its neighbours.

Parameterized verification of broadcast networks amounts to checking a given property independently of the initial configuration, and in particular independently of the number of agents and communication topology. A natural property one can be interested in is coverability: a state of the broadcast protocol is coverable if some execution leads to a configuration in which one node is in that local state. When considering error states, a positive instance for the coverability problem thus corresponds to a network that can exhibit a bad behaviour.

Coverability is undecidable for static broadcast networks [3], *i.e.* when the communication topology is fixed along executions. Decidability can be recovered by relaxing the semantics and allowing non-deterministic reconfigurations of the communication topology. In reconfigurable broadcast networks, coverability of a control state is decidable in PTIME [2]. A simple saturation algorithm allows to compute the set of all states of the broadcast protocol that can be covered.

Cutoff and covering length. Two important characteristics of positive instances of the coverability problem are the cutoff and the covering length. First, the *cutoff* is the minimal number of agents for which a covering execution exists. The notion of cutoff is particularly relevant for reconfigurable broadcast networks since they enjoy a monotonicity property: if a state can be covered from a configuration, it can also be from any configuration with more nodes. Second, the *covering length* is the minimal number of steps for covering executions. It weighs how fast a network execution can go wrong. Both the cutoff and the covering length are somehow complexity measures for the coverability problem. Surprisingly, no upper nor lower bounds on these values appear in the literature for reconfigurable broadcast networks.

Contributions. In this paper, we prove a tight linear bound for the cutoff, and a tight quadratic bound for the covering length in reconfigurable broadcast networks. Both are expressed in the number of states of the broadcast protocol. These are obtained by refining the saturation algorithm that computes the set of coverable states, and finely analysing it.

Another contribution is to introduce lossy broadcast networks, in which the communication topology is fixed, however errors in message transmission may occur. In contrast with broadcast networks with losses that appear in the literature [4], in our model, message losses happen upon sending, rather than upon reception. This makes a crucial difference: reconfiguration of the communication topology can easily be encoded by losses upon reception, whereas it is not obvious for losses upon sending. Perhaps surprisingly, we prove that the set of states that can be covered in reconfigurable semantics agrees with the one in static lossy semantics. Using the same refined saturation algorithm, we prove that same tight bounds hold for lossy broadcast networks: the cutoff is linear, and the covering length is quadratic (in the number of states of the broadcast protocol). The two semantics thus appear quite similar, yet, we show that the reconfigurable semantics can be linearly more succinct (in terms of number of nodes) than the lossy semantics.

Finally, we study a natural decision problem related to the cutoff: decide whether a state is coverable (in either semantics) with a fixed number of nodes. We prove it to be NP-complete.

Outline. In Section 2, we define the broadcast networks, with static, reconfiurable and lossy semantics. In Section 3, we present our tight bounds for cutoff and covering length. In Section 4, we show our succinctness result. In Section 5, we give our NP-completeness result.

2 Broadcast networks

2.1 Static broadcast networks

▶ **Definition 1.** *A* broadcast protocol *is a tuple* $\mathcal{P} = (Q, I, \Sigma, \Delta)$ *where* Q *is a finite set of control states;* $I \subseteq Q$ *is the set of initial control states;* Σ *is a finite alphabet; and* $\Delta \subseteq (Q \times \{!a, ?a \mid a \in \Sigma\} \times Q)$ *is the transition relation.*

For ease of readability, we often write $q \xrightarrow{!a} q'$ (resp. $q \xrightarrow{?a} q'$) for $(q, !a, q') \in \Delta$ (resp. $(q, ?a, q') \in \Delta$). We assume all broadcast networks to be complete for receptions: for every $q \in Q$ and $a \in \Sigma$, there exists q' such that $q \xrightarrow{?a} q'$.

A broadcast protocol is represented in Figure 1. In this example and in the whole paper, for concision purposes, we assume that if the reception of a message is unspecified from some state, it implicitly represents a self-loop. For example here, from q_1, receiving a leads to q_1 again.

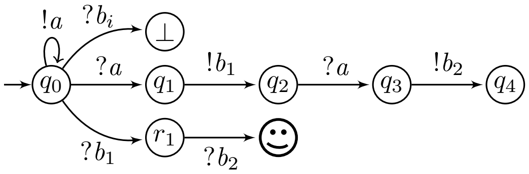


Figure 1 Example of a broadcast protocol.

Broadcast networks involve several copies, or *nodes*, of the same broadcast protocol $\mathcal{P}$. A configuration is an undirected graph whose vertices are labelled with a state of Q. Transitions between configurations happen by broadcasts from a node to its neighbours.

Formally, given a broadcast protocol $\mathcal{P} = (Q, I, \Sigma, \Delta)$, a *configuration* is an undirected graph $\gamma = (\mathsf{N}, \mathsf{E}, \mathsf{L})$ where N is a finite set of nodes; $\mathsf{E} \subseteq \mathsf{N} \times \mathsf{N}$ is a symmetric and irreflexive relation describing the set of edges; finally, $\mathsf{L}\colon \mathsf{N} \to Q$ is the labelling function. We let $\Gamma(\mathcal{P})$ denote the (infinite) set of Q-labelled graphs. Given a configuration $\gamma \in \Gamma(\mathcal{P})$, we write $\mathsf{n} \sim \mathsf{n}'$ whenever $(\mathsf{n}, \mathsf{n}') \in \mathsf{E}$ and we let $\mathsf{Neigh}_\gamma(\mathsf{n}) = \{\mathsf{n}' \in \mathsf{N} \mid \mathsf{n} \sim \mathsf{n}'\}$ be the neighbourhood of n, *i.e.* the set of nodes adjacent to n. Finally $\mathsf{L}(\gamma)$ denotes the set of labels appearing in nodes of γ. A configuration $(\mathsf{N}, \mathsf{E}, \mathsf{L})$ is called *initial* if $\mathsf{L}(\mathsf{N}) \subseteq I$.

The operational semantics of a static broadcast network for a given broadcast protocol $\mathcal{P}$ is an infinite-state transition system $\mathcal{T}(\mathcal{P})$. Intuitively, each node of a configuration runs protocol $\mathcal{P}$, and may send/receive messages to/from its neighbours. From a configuration $\gamma = (\mathsf{N}, \mathsf{E}, \mathsf{L})$, there is a step to $\gamma' = (\mathsf{N}', \mathsf{E}', \mathsf{L}')$ if $\mathsf{N}' = \mathsf{N}$, $\mathsf{E}' = \mathsf{E}$, and there exists $\mathsf{n} \in \mathsf{N}$ and $a \in \Sigma$ such that $(\mathsf{L}(\mathsf{n}), !a, \mathsf{L}'(\mathsf{n})) \in \Delta$, and for every $\mathsf{n}' \in \mathsf{N}$, if $\mathsf{n}' \in \mathsf{Neigh}_\gamma(\mathsf{n})$, then

$(\mathsf{L}(\mathsf{n}'), ?a, \mathsf{L}'(\mathsf{n}')) \in \Delta$, otherwise $\mathsf{L}'(\mathsf{n}') = \mathsf{L}(\mathsf{n}')$: a step reflects how nodes evolve when one of them broadcasts a message to its neighbours. We write $\gamma \xrightarrow{\mathsf{n},!a}_{\mathtt{s}} \gamma'$, or simply $\gamma \rightarrow_{\mathtt{s}} \gamma'$ (the $\mathtt{s}$ subscript emphasizes that the communication topology is *static*).

An *execution* of the static broadcast network is a sequence $\rho = (\gamma_i)_{0 \le i \le r}$ of configurations $(\mathsf{N}, \mathsf{E}, \mathsf{L}_i)$ such that γ_0 is an initial configuration, and for every $0 \le i < r$, $\gamma_i \rightarrow_{\mathtt{s}} \gamma_{i+1}$. We write $\#\mathsf{nodes}(\rho)$ for the number of nodes in γ_0, $\#\mathsf{steps}(\rho)$ for the number r of steps along ρ, and for any node $\mathsf{n} \in \mathsf{N}$, $\#\mathsf{steps}(\rho, \mathsf{n})$ for the number of broadcasts, called the *active length*, of node n along ρ. Note that, along an execution, the number of nodes and the communication topology are fixed. The set of all static executions is denoted $\mathsf{Exec}_{\mathtt{s}}(\mathcal{P})$.

Coverability problem

Given a broadcast protocol $\mathcal{P}$ and a subset of target states $F \subseteq Q$, we write $\mathsf{COVER}_{\mathtt{s}}(\mathcal{P}, F)$ for the set of all *covering* executions, that is, executions that visit a configuration with a node labelled by a state in F:

$$\mathsf{COVER}_{\mathtt{s}}(\mathcal{P}, F) = \{(\gamma_i)_{0 \le i \le r} \in \mathsf{Exec}_{\mathtt{s}}(\mathcal{P}) \mid \mathsf{L}(\gamma_r) \cap F \neq \emptyset\}.$$

The *coverability problem* is a decision problem that takes a broadcast protocol $\mathcal{P}$ and a subset of target states F as inputs, and outputs whether $\mathsf{COVER}_{\mathtt{s}}(\mathcal{P}, F)$ is nonempty. For broadcast networks, the coverability problem is a parameterized verification problem, since the number of initial configurations is infinite. It is known that coverability is undecidable for static broadcast networks [3], since one can use the communication topology to build chains that may encode values of counters, and hence simulate Minsky machines [10].

If the broadcast protocol $\mathcal{P}$ allows to cover the subset F, we define the *cutoff* as the minimal number of nodes required in an execution to cover F. Similarly, we define the *covering length* as the length of a shortest finite execution covering F. Those values are important to characterize the complexity of a broadcast protocol: assuming a safe set of states, coverability of the complement set represents bad behaviours, and cutoff and covering length measure the size of minimal witnesses for violation of the safety property.

2.2 Reconfigurable broadcast networks

To circumvent the undecidability of coverability for static broadcast networks, one attempt is to introduce non-deterministic reconfiguration of the communication topology. This solution also allows one to model arbitrary mobility of the nodes, which is meaningful, *e.g.* for mobile ad-hoc networks [3].

Under this semantics, configurations are the same as under the static semantics. Transitions between configurations however are enhanced by the ability to modify the communication topology before performing a broadcast. Formally, from a configuration $\gamma = (\mathsf{N}, \mathsf{E}, \mathsf{L})$, there is a step to $\gamma' = (\mathsf{N}', \mathsf{E}', \mathsf{L}')$ if $\mathsf{N}' = \mathsf{N}$, and there exists $\mathsf{n} \in \mathsf{N}$ and $a \in \Sigma$ such that $(\mathsf{L}(\mathsf{n}), !a, \mathsf{L}'(\mathsf{n})) \in \Delta$, and for every $\mathsf{n}' \in \mathsf{N}$, if $\mathsf{n}' \in \mathsf{Neigh}_{\gamma'}(\mathsf{n})$, then $(\mathsf{L}(\mathsf{n}), ?a, \mathsf{L}'(\mathsf{n}')) \in \Delta$, otherwise $\mathsf{L}'(\mathsf{n}') = \mathsf{L}(\mathsf{n}')$: a step thus reflects that the communication topology may change from E to E' followed by the broadcast of a message from a node to its neighbours in the new topology. We write $\gamma \xrightarrow{\mathsf{n},!a}_{\mathtt{r}} \gamma'$, or simply $\gamma \rightarrow_{\mathtt{r}} \gamma'$.

Similarly to the static case, we write $\mathsf{Exec}_{\mathtt{r}}(\mathcal{P})$ and $\mathsf{COVER}_{\mathtt{r}}(\mathcal{P}, F)$ for, respectively the set of all reconfigurable executions in $\mathcal{P}$, and the set of all reconfigurable executions in $\mathcal{P}$ that cover F. We will also use the same notations $\#\mathsf{nodes}(\rho)$, $\#\mathsf{steps}(\rho)$ and $\#\mathsf{steps}(\rho, \mathsf{n})$ as in the static case.

Figure 7 (with $n = 2$) gives an example of reconfigurable execution for the broadcast protocol of Figure 1 (which covers ☺). Note that the communication topology indeed evolves along the execution. Here the colored nodes broadcast a message in the step leading to the next configuration.

A noticeable property of reconfigurable broadcast networks is the following copycat property. Such a monotonicity property was originally shown in [7] for asynchronous shared-memory systems, and it also applies to our context.

▶ **Proposition 2** (Copycat for reconfigurable semantics). *Given $\rho : \gamma_0 \rightarrow_{\mathtt{r}} \gamma_1 \cdots \rightarrow_{\mathtt{r}} \gamma_s$ an execution, with $\gamma_s = (\mathsf{N}, \mathsf{E}, \mathsf{L})$, for every $q \in \mathsf{L}(\gamma_s)$, for every $\mathsf{n}^q \in \mathsf{N}$ such that $\mathsf{L}(\mathsf{n}^q) = q$, there exists $t \in \mathbb{N}$ and an execution $\rho' : \gamma'_0 \rightarrow_{\mathtt{r}} \gamma'_1 \cdots \rightarrow_{\mathtt{r}} \gamma'_t$ with $\gamma'_t = (\mathsf{N}', \mathsf{E}', \mathsf{L}')$ such that $|\mathsf{N}'| = |\mathsf{N}|+1$, there is an injection $\iota : \mathsf{N} \rightarrow \mathsf{N}'$ with for every $\mathsf{n} \in \mathsf{N}$, $\mathsf{L}'(\iota(\mathsf{n})) = \mathsf{L}(\mathsf{n})$, and for the extra node $\mathsf{n}_{\mathsf{fresh}} \in \mathsf{N}' \setminus \iota(\mathsf{N})$, $\mathsf{L}'(\mathsf{n}_{\mathsf{fresh}}) = q$, and $\#\mathsf{steps}(\rho', \mathsf{n}_{\mathsf{fresh}}) = \#\mathsf{steps}(\rho, \mathsf{n}^q)$.*

Intuitively, the new node $\mathsf{n}_{\mathsf{fresh}}$ will copy the moves of node n^q: it performs the same broadcasts (but to nobody) and receives the same messages. More precisely, when n^q broadcasts in ρ, it does so also in ρ' and then we disconnect all the nodes and $\mathsf{n}_{\mathsf{fresh}}$ repeats the broadcast (no other node is affected because of the disconnection); when n^q receives a message in ρ, we connect $\mathsf{n}_{\mathsf{fresh}}$ to the same neighbours as n^q (*i.e.*, $\iota(\mathsf{n}) \sim' \mathsf{n}_{\mathsf{fresh}}$ if and only if $\mathsf{n} \sim \mathsf{n}^q$) so that $\mathsf{n}_{\mathsf{fresh}}$ also receives the same message in ρ'.

Relying on the copycat property, when reconfigurations are allowed, the coverability problem becomes decidable and solvable in polynomial time.

▶ **Theorem 3** ([2]). *Coverability is decidable in* PTIME *for reconfigurable broadcast networks.*

More precisely, a simple saturation algorithm allows one to compute in polynomial time, the set of all states that can be covered. Despite this complexity result, to the best of our knowledge, no bounds on the cutoff or length of witness executions are stated in the literature.

2.3 Broadcast networks with messages losses

Communication failures were studied for broadcast networks, assuming non-deterministic message losses could happen: when a message is broadcast, some of the neighbours of the sending node may not receive it [4]. As observed by the authors, the coverability problem for such networks easily reduces to the coverability problem in reconfigurable networks by considering a complete topology, and message losses are simulated by reconfigurations. Thus, message losses upon reception are equivalent to reconfiguration of the communication topology.

We propose an alternative semantics here: when a message is broadcast, it either reaches all neighbours of the sending node, or none of them. This is relevant in contexts where broadcasts are performed in an atomic manner and may fail. In contrast to message losses upon reception, it is not obvious to simulate arbitrary reconfigurations of the communication topology with such message losses.

Formally, from a configuration $\gamma = (\mathsf{N}, \mathsf{E}, \mathsf{L})$, there is a step to $\gamma' = (\mathsf{N}', \mathsf{E}', \mathsf{L}')$ if $\mathsf{N}' = \mathsf{N}$, $\mathsf{E}' = \mathsf{E}$ and there exists $\mathsf{n} \in \mathsf{N}$ and $a \in \Sigma$ such that $(\mathsf{L}(\mathsf{n}), !a, \mathsf{L}'(\mathsf{n})) \in \Delta$, and either (a) for every $\mathsf{n}' \neq \mathsf{n}$, $\mathsf{L}'(\mathsf{n}') = \mathsf{L}(\mathsf{n}')$ (no one has received the message, it has been lost), or (b) if $\mathsf{n}' \in \mathsf{Neigh}_{\gamma'}(\mathsf{n})$, then $(\mathsf{L}(\mathsf{n}'), ?a, \mathsf{L}'(\mathsf{n}')) \in \Delta$, otherwise $\mathsf{L}'(\mathsf{n}') = \mathsf{L}(\mathsf{n}')$: a step thus reflects that the broadcast message may be lost when it is sent. We write $\gamma \xrightarrow{\mathsf{n},!a}_{\mathtt{l}} \gamma'$ or simply $\gamma \rightarrow_{\mathtt{l}} \gamma'$. Similarly to the static and reconfigurable semantics, $\#\mathsf{steps}(\rho, \mathsf{n})$ is the number of broadcasts (including lost ones) by node n along ρ; and we write $\#\mathsf{nonlost\_steps}(\rho, \mathsf{n})$ for the number of successful broadcasts by node n along ρ.

For lossy executions also, we use the following notations: $\mathsf{Exec}_{\mathtt{l}}(\mathcal{P})$ and $\mathsf{COVER}_{\mathtt{l}}(\mathcal{P}, F)$. Any lossy execution can be seen as a reconfigurable execution. Indeed, a lossy execution with communication topology E can be transformed into a reconfigurable one in which the communication topology of each configuration is either $\emptyset$ or E, depending on whether the next broadcast is lost or not. Therefore, with slight abuse of notation, we write $\mathsf{Exec}_{\mathtt{l}}(\mathcal{P}) \subseteq \mathsf{Exec}_{\mathtt{r}}(\mathcal{P})$.

Figure 2 gives an example of a lossy execution for the broadcast protocol of Figure 1. Note that in the third transition, some node indeed performs a lossy broadcast, emphasized by the subscript "lost". As before, the colored nodes broadcast a message in the step leading to the next configuration.

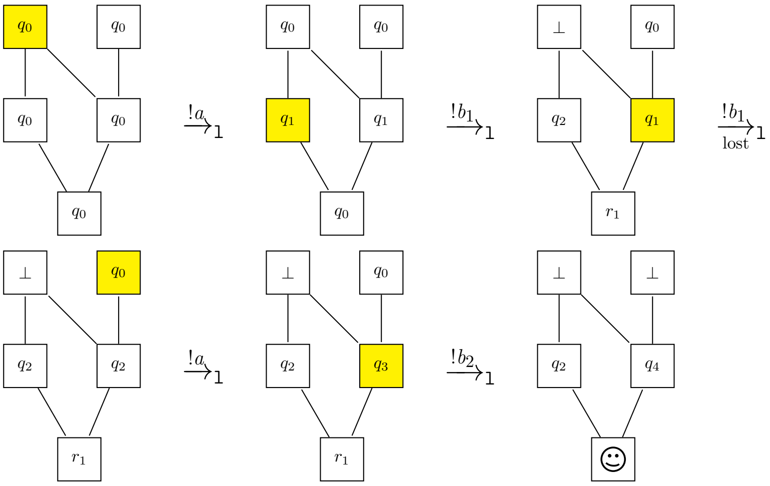


Figure 2 Example of a lossy execution on the protocol from Figure 1.

3 Tight bounds for reconfigurable and lossy broadcast networks

In this section, we will show tight bounds for the cutoff and the minimal length of a witness execution for the coverability problem. These hold both for the reconfigurable and the lossy semantics.

3.1 Upper bounds on cutoff and covering length for reconfigurable networks

First, we will refine the polytime saturation algorithm of [2], which computes all states which can be covered in the reconfigurable semantics. We will then show that, based on the underlying computation, one can construct small witnesses for the two semantics (linear number of nodes and quadratic number of steps). While it would be enough to show the result for the lossy semantics (since, given a broadcast protocol $\mathcal{P}$, $\mathsf{Exec}_{\mathtt{l}}(\mathcal{P}) \subseteq \mathsf{Exec}_{\mathtt{r}}(\mathcal{P})$), for pedagogical reasons, we provide the two proofs, starting with the simplest one for reconfigurable semantics.

Let us fix for the rest of this section, a protocol $\mathcal{P} = (Q, I, \Sigma, \Delta)$. We slightly modify the algorithm given in [2] as follows: we include at most one state to the set S in each iteration. Additionally, we associate a labelling function $c : S \to \mathbb{N}$ with the set S in every iteration. More formally, we consider the modification of the previous saturation algorithm as shown in Algorithm 1.

Algorithm 1 Refined saturation algorithm for coverability.

```
$S := I$; $c(S) := |I|$; $S' := \emptyset$
while $S \neq S'$ do
    $S' := S$; $c := c(S)$
    if $\exists (q_1, !a, q_2) \in \Delta$ s.t. $q_1 \in S'$ and $q_2 \notin S'$ then
        $S := S \cup \{q_2\}$; $c(S) := c + 1$
    else if $\exists (q_1, !a, q_2) \in \Delta$ and $(q_1', ?a, q_2') \in \Delta$ s.t. $q_1, q_2, q_1' \in S'$ and $q_2' \notin S'$ then
        $S := S \cup \{q_2'\}$; $c(S) := c + 2$
    end if
end while
return $S$
```

In Algorithm 1, the variable c counts the number of nodes that are sufficient to cover the current set S, as we will prove later.

▶ **Lemma 4** ([2]). *Algorithm 1 terminates and returns the set of coverable states. In particular,* $\mathsf{COVER}_{\mathtt{r}}(\mathcal{P}, F) \neq \emptyset$ *iff* $F \cap S \neq \emptyset$.

Let $S_0, S_1, \ldots, S_m$ be the sets after each iteration of the algorithm, with $S_0 = I$ and $S_m = S$. We fix an ordering on the states in S on the basis of insertion in S: for all $1 \leq i \leq m$, q_i is such that $q_i \in S_i \setminus S_{i-1}$. In the following, we show the desired upper bounds, proving that there exists an execution of size $O(n)$ and length $O(n^2)$ covering at the same time all states of S_m.

▶ **Theorem 5.** *Let* $\mathcal{P} = (Q, I, \Sigma, \Delta)$ *be a broadcast protocol, and* $F \subseteq Q$. *If* $\mathsf{COVER}_{\mathtt{r}}(\mathcal{P}, F) \neq \emptyset$ *(that is, if* $F \cap S \neq \emptyset$*), then there exists* $\rho \in \mathsf{COVER}_{\mathtt{r}}(\mathcal{P}, F)$ *with* $\#\mathsf{nodes}(\rho) \leq 2|Q|$ *and* $\#\mathsf{steps}(\rho) \leq 2|Q|^2$.

Theorem 5 is a consequence of the following Lemma.

▶ **Lemma 6.** *For every step* i *of Algorithm 1, there exists an initial configuration* γ_0*, a configuration* γ *and a reconfigurable execution* $\rho : \gamma_0 \xrightarrow{*}_{\mathtt{r}} \gamma$ *such that* $\mathsf{L}(\gamma) = S_i$, $\#\mathsf{nodes}(\rho) = c(S_i)$*, and* $\max_{\mathsf{n}} \#\mathsf{steps}(\rho, \mathsf{n}) \leq i$.

Proof. The lemma is proved by induction on i. The base case $i = 0$ is obvious: take the initial configuration γ_0 with $|I|$ nodes, and label each node with a different initial state; its size is $|I|$, and the length of the execution is 0, hence so is the maximum active length.

To prove the induction step, we distinguish two cases: depending on whether q_{i+1} was added as the target state of a broadcast transition $q \xrightarrow{!a}$ for some $q \in S_i$; or whether q_{i+1} is the target state of a reception from some $q \in S_i$ with matching broadcast between two states already in S_i.

Case 1: There exists $q \in S_i$ with $q \xrightarrow{!a} q_{i+1}$. We apply the induction hypothesis to step i, and exhibit an execution $\rho : \gamma_0 \xrightarrow{*}_{\mathtt{r}} \gamma$ such that $\mathsf{L}(\gamma) = S_i$, $\#\mathsf{nodes}(\rho) = c(S_i)$ and $\max_{\mathsf{n}} \#\mathsf{steps}(\rho, \mathsf{n}) \leq i$. Applying the copycat property (see Proposition 2), we construct an execution $\rho' : \gamma_0' \xrightarrow{*}_{\mathtt{r}} \gamma'$ such that γ_0' has one node more than γ_0, and, focusing on the nodes (since we are in a reconfigurable setting, edges in the configuration are not important), γ' coincides with γ, with an extra node n labelled by q. We then disconnect all nodes and extend with a transition $\gamma' \xrightarrow{\mathsf{n}, !a}_{\mathtt{r}} \gamma''$, which makes only progress node n from q to q_{i+1}; the resulting execution is denoted ρ''. Then:

1. $\mathsf{L}(\gamma'') = S_i \cup \{q_{i+1}\} = S_{i+1}$,
2. $\#\mathsf{nodes}(\rho'') = c(S_i) + 1 = c(S_{i+1})$,
3. $\max_{\mathsf{n}} \#\mathsf{steps}(\rho'', \mathsf{n}) \leq \max_{\mathsf{n}} \#\mathsf{steps}(\rho, \mathsf{n}) + 1 \leq i + 1$; Indeed, the active length of the copycat node along ρ' coincides with the active length of some existing node along ρ, and it is increased only by 1 in ρ''.

This proves the induction step in the first case.

Case 2: There exists $q, q', q'' \in S_i$ with $q \xrightarrow{?a} q_{i+1}$ and $q' \xrightarrow{!a} q''$. The idea is similar to the previous case, but one should apply the copycat property twice, to both q and q'. We formalize this.

We apply the induction hypothesis to step i, and exhibit an execution $\rho : \gamma_0 \xrightarrow{*}_{\mathtt{r}} \gamma$ such that $\mathsf{L}(\gamma) = S_i$, $\#\mathsf{nodes}(\rho) = c(S_i)$ and $\max_{\mathsf{n}} \#\mathsf{steps}(\rho, \mathsf{n}) \leq i$. Applying the copycat property (see Proposition 2) twice, to both q and q', we construct an execution $\rho' : \gamma'_0 \xrightarrow{*}_{\mathtt{r}} \gamma'$ such that γ'_0 has two nodes more than γ_0, and, focusing on the nodes, γ' coincides with γ, with one extra node n labelled by q and one extra node n' labelled by q'. We then connect nodes n and n' and disconnect all other nodes, and extend with a transition $\gamma' \xrightarrow{\mathsf{n}',!a}_{\mathtt{r}} \gamma''$; this makes node n progress from q to q_{i+1} and node n' progress from q' to q''; all other nodes are unchanged; the resulting execution is denoted ρ''. Then:

1. $\mathsf{L}(\gamma'') = S_i \cup \{q'', q_{i+1}\} = S_{i+1}$ since $q'' \in S_i$,
2. $\#\mathsf{nodes}(\rho'') = c(S_i) + 2 = c(S_{i+1})$,
3. $\max_{\mathsf{n}} \#\mathsf{steps}(\rho'', \mathsf{n}) \leq \max_{\mathsf{n}} \#\mathsf{steps}(\rho, \mathsf{n}) + 1 \leq i + 1$; Indeed the active length of any of the copycat node along ρ' coincides with the active length of some existing node along ρ, and it is increased by at most 1 in ρ''.

This proves the induction step in the second case, which allows to conclude the proof of the lemma. ◀

To conclude the proof of Theorem 5, we recall that Algorithm 1 is sound and complete: S_m is the set of states that can be covered. Hence, from Lemma 6, we deduce that if $\mathsf{COVER}_{\mathtt{r}}(\mathcal{P}, F) \neq \emptyset$, then there is $\rho \in \mathsf{COVER}_{\mathtt{r}}(\mathcal{P}, F)$ such that:

1. $\mathsf{L}(\gamma) = S_m$;
2. $\#\mathsf{nodes}(\rho) = c(S_m) \leq |I| + 2m \leq |I| + 2(|Q| - |I|) = 2|Q| - |I|$;
3. $\max_{\mathsf{n}} \#\mathsf{steps}(\rho, \mathsf{n}) \leq m \leq |Q| - |I|$.

Therefore $\#\mathsf{steps}(\rho) \leq (\#\mathsf{nodes}(\rho)) \cdot \Big(\max_{\mathsf{n}} \#\mathsf{steps}(\rho, \mathsf{n})\Big) \leq 2|Q|^2$, so that we established the desired bounds for Theorem 5.

3.2 Upper bounds on cutoff and covering length for lossy networks

Perhaps surprisingly, Algorithm 1 also computes the set of states that can be covered by lossy executions. Concerning coverable states, the reconfigurable and lossy semantics thus agree. In Section 4, we will show that reconfigurable covering executions can be linearly more succinct than lossy covering executions.

▶ **Lemma 7.** *Algorithm 1 returns the set of coverable states for lossy broadcast networks. In particular,* $\mathsf{COVER}_{\mathtt{l}}(\mathcal{P}, F) \neq \emptyset$ *iff* $F \cap S \neq \emptyset$.

Indeed, we have $\mathsf{Exec}_{\mathtt{l}}(\mathcal{P}) \subseteq \mathsf{Exec}_{\mathtt{r}}(\mathcal{P})$. Therefore $\mathsf{COVER}_{\mathtt{l}}(\mathcal{P}, F) \neq \emptyset$ implies $\mathsf{COVER}_{\mathtt{r}}(\mathcal{P}, F) \neq \emptyset$ and by Lemma 4, we conclude $F \cap S \neq \emptyset$. The other direction of Lemma 7 is a consequence of the following theorem.

▶ **Theorem 8.** *Let* $\mathcal{P} = (Q, I, \Sigma, \Delta)$ *be a broadcast protocol, and* $F \subseteq Q$. *If* $S \cap F \neq \emptyset$, *then there exists* $\rho \in \mathsf{COVER}_{\mathtt{l}}(\mathcal{P}, F)$ *with* $\#\mathsf{nodes}(\rho) \leq 2|Q|$ *and* $\#\mathsf{steps}(\rho) \leq 2|Q|^2$.

Before going to the proof of Theorem 8, we show a copycat property for the lossy broadcast networks, as a counterpart of Proposition 2 for the lossy semantics. Since the communication topology is static in lossy networks, the following proposition explicitly relates the communication topologies in the initial execution and its copycat extension.

▶ **Proposition 9** (Copycat for lossy semantics). *Given $\rho : \gamma_0 \rightarrow_1 \gamma_1 \cdots \rightarrow_1 \gamma_r$ an execution, with $\gamma_r = (\mathsf{N}, \mathsf{E}, \mathsf{L})$, for every $q \in \mathsf{L}(\gamma_r)$, for every $\mathsf{n}^q \in \mathsf{N}$ such that $\mathsf{L}(\mathsf{n}^q) = q$, there exists $s \in \mathbb{N}$ and an execution $\rho' : \gamma'_0 \rightarrow_1 \gamma'_1 \cdots \rightarrow_1 \gamma'_s$ with $\gamma'_s = (\mathsf{N}', \mathsf{E}', \mathsf{L}')$ such that $|\mathsf{N}'| = |\mathsf{N}|+1$, there is an injection $\iota : \mathsf{N} \to \mathsf{N}'$ with for every $\mathsf{n} \in \mathsf{N}$, $\mathsf{L}'(\iota(\mathsf{n})) = \mathsf{L}(\mathsf{n})$, and for the extra node $\mathsf{n}_{\mathsf{fresh}} \in \mathsf{N}' \setminus \iota(\mathsf{N})$, $\mathsf{L}'(\mathsf{n}_{\mathsf{fresh}}) = q$, for every $\mathsf{n} \in \mathsf{N}$, $\mathsf{n}_{\mathsf{fresh}} \sim' \iota(\mathsf{n})$ iff $\mathsf{n}^q \sim \mathsf{n}$, $\#\mathsf{steps}(\rho', \mathsf{n}_{\mathsf{fresh}}) = \#\mathsf{steps}(\rho, \mathsf{n}^q)$, and $\#\mathsf{nonlost\_steps}(\rho', \mathsf{n}_{\mathsf{fresh}}) = 0$.*

Proof. First notice that, from our definition of lossy semantics, the topology should be the same in γ_0 and in γ_r, hence we can write $\gamma_0 = (\mathsf{N}, \mathsf{E}, \mathsf{L}_0)$, and more generally, for every i, $\gamma_i = (\mathsf{N}, \mathsf{E}, \mathsf{L}_i)$. Define N' as a finite set such that $|\mathsf{N}'| = |\mathsf{N}| + 1$, and fix an injection $\iota : \mathsf{N} \to \mathsf{N}'$. Write $\mathsf{n}_{\mathsf{fresh}}$ for the unique element of $\mathsf{N}' \setminus \iota(\mathsf{N})$. Set $\mathsf{L}'_0(\iota(\mathsf{n})) = \mathsf{L}_0(\mathsf{n})$ for every $\mathsf{n} \in \mathsf{N}$, and $\mathsf{L}'_0(\mathsf{n}_{\mathsf{fresh}}) = \mathsf{L}_0(\mathsf{n}^q)$. Define the edge relation E' by its induced edge relation $\sim'$ such that $\iota(\mathsf{n}) \sim' \iota(\mathsf{n}')$ iff $\mathsf{n} \sim \mathsf{n}'$, and $\mathsf{n}_{\mathsf{fresh}} \sim' \iota(\mathsf{n}')$ iff $\mathsf{n}^q \sim \mathsf{n}'$.

The idea will then be to make $\mathsf{n}_{\mathsf{fresh}}$ follow what n^q is doing. Roughly, if n^q is receiving a message to progress, then we will connect $\mathsf{n}_{\mathsf{fresh}}$ to a relevant node to also receive the message; if n^q is broadcasting a message, then we will make $\mathsf{n}_{\mathsf{fresh}}$ broadcast a message and lose, so that no other node is impacted.

Formally, we will show by induction on i that for every $0 \leq i \leq r$, there is an execution $\rho'_i : \gamma'_0 \rightarrow_1 \gamma'_1 \cdots \rightarrow_1 \gamma'_{f(i)}$ for some $f(i)$, such that $\mathsf{L}'_i(\iota(\mathsf{n})) = \mathsf{L}_i(\mathsf{n})$ for every $\mathsf{n} \in \mathsf{N}$ and $\mathsf{L}'_i(\mathsf{n}_{\mathsf{fresh}}) = \mathsf{L}_i(\mathsf{n}^q)$. The initial case $i = 0$ is obvious. We then assume that we have constructed a relevant ρ'_i for some $i < r$, and we will extend it to ρ'_{i+1} as follows. We make a case distinction depending on the nature of the step $\gamma_i \rightarrow_1 \gamma_{i+1}$:

- Assume $\gamma_i \xrightarrow{\mathsf{n},!a}_1 \gamma_{i+1}$ is a broadcast message with $\mathsf{n}^q \neq \mathsf{n}$, then ρ'_{i+1} is obtained by extending ρ'_i with the broadcast $\gamma'_{f(i)} \xrightarrow{\iota(\mathsf{n}),!a} \gamma'_{f(i)+1}$, with the condition that it should be lost if and only if it was lost in the original execution. For checking correctness, we distinguish two cases:
  - the broadcast message was not lost, and $\mathsf{n}^q \sim \mathsf{n}$. Then, it is the case that $\mathsf{n}_{\mathsf{fresh}} \sim' \iota(\mathsf{n})$, hence $\mathsf{n}_{\mathsf{fresh}}$ also receives the message. By resolving properly the nondeterminism, we can make the label of $\mathsf{n}_{\mathsf{fresh}}$ become the same as the label of n^q in $\gamma'_{f(i)+1}$. Note also that all nodes in $\iota(\mathsf{N})$ can progress to the same states as those of N in γ_{i+1};
  - the broadcast message was lost, or $\mathsf{n}^q \not\sim \mathsf{n}$, then it is the case that the label of n^q has not been changed in $\gamma_i \xrightarrow{\mathsf{n},!a}_1 \gamma_{i+1}$, and so will the label of the fresh node in $\gamma'_{f(i)}$.
- Assume $\gamma_i \xrightarrow{\mathsf{n}^q,!a}_1 \gamma_{i+1}$ is a broadcast message, then we extend ρ'_i with the two steps $\gamma'_{f(i)} \xrightarrow{\iota(\mathsf{n}^q),!a} \gamma'_{f(i)+1} \xrightarrow{\mathsf{n}_{\mathsf{fresh}},!a} \gamma'_{f(i)+2}$ (resolving nondeterminism in a similar way as in $\gamma_i \xrightarrow{\mathsf{n}^q,!a}_1 \gamma_{i+1}$), and we make the last broadcast lossy whereas the broadcast from $\iota(\mathsf{n}^q)$ is lossy if and only if it was lossy in $\gamma_i \rightarrow_1 \gamma_{i+1}$.

This concludes the induction. Notice that in the constructed execution, node $\mathsf{n}_{\mathsf{fresh}}$ does not make any real sending. ◀

For any configuration $\gamma = (\mathsf{N}, \mathsf{E}, \mathsf{L})$ and a node n, we write $\mathsf{L}(\mathsf{n}) = \times$ if n is not important anymore in the execution, in other words all the required conditions in γ' such that $\gamma \xrightarrow{*}_1 \gamma'$ are still satisfied whatever $\mathsf{L}(\mathsf{n})$ is.

Recall the saturation algorithm and the ordering of the sets and the states: $S_0 = I, S_1, \ldots, S_m = S$ are the sets after each iteration and q_i is the state such that $q_i \in S_i \setminus S_{i-1}$ for all $1 \leq i \leq m$. We will refine the construction from the proof of Lemma 6 (in the context of reconfigurable broadcast networks), and build inductively a lossy execution covering all states in S_i. Since the topology is static, some nodes which have "finished their jobs" will remain connected to other nodes, and may therefore continue to change states (contrary to Lemma 6 where they could be fully disconnected). Hence, in every such execution, every state $q \in S_i$ (which is then covered by the execution) will have a main corresponding node, whose label will remain q. All nodes which are not the main node of a state will be assigned $\times$, since their labels will become meaningless.

We formalize this idea in the lemma below. However, for better understanding, we also illustrate this inductive construction of a witness execution in Figure 4 on the simple broadcast protocol from Figure 3. Configurations are represented vertically: they involve 10 nodes, and the communication topology is given for the first configuration only, for the sake of readability. To save space, several broadcasts (of the same message type, from different nodes) may happen in a *macrostep* that merges several steps. This is for instance the case in the first macrostep, where a is being broadcast from the node in set S_1, as well as from the first node in set S_2. Dashed arrows are used to represent that a node is not involved in some macrostep and thus stays in the same state. In the execution, the nodes that are performing a real broadcast are colored yellow, the ones which receive a message are colored gray, and blue nodes indicate the main nodes for the coverable states.

▶ **Lemma 10.** *For every step i of the refined saturation algorithm, there exists a configuration $\gamma = (\mathsf{N}, \mathsf{E}, \mathsf{L})$ and an execution $\rho : \gamma_0 \xrightarrow{*}_1 \gamma$ such that:*

- $\mathsf{L}(\gamma) \setminus \{\times\} = S_i$ *and* $\#\mathsf{nodes}(\rho) = c(S_i)$*,*
- $\max_{\mathsf{n}} \#\mathsf{steps}(\rho, \mathsf{n}) \leq i$ *and* $\max_{\mathsf{n}} \#\mathsf{nonlost\_steps}(\rho, \mathsf{n}) \leq 1$*,*
- *for every $q \in S_i$, there exists $\mathsf{n}_q^{\mathsf{main}} \in \mathsf{N}$ such that*
  - $\mathsf{L}(\mathsf{n}_q^{\mathsf{main}}) = q$ *and* $\#\mathsf{nonlost\_steps}(\rho, \mathsf{n}_q^{\mathsf{main}}) = 0$*,*
  - $\mathsf{n}_q^{\mathsf{main}} \sim \mathsf{n}$ *implies* $\mathsf{L}(\mathsf{n}) = \times$*, and if* $\mathsf{n} \notin \{\mathsf{n}_q^{\mathsf{main}} \mid q \in S_i\}$*, then* $\mathsf{L}(\mathsf{n}) = \times$*.*

Proof. We do the proof by induction on i. The case $i = 0$ is obvious, by picking one main node per initial state in I, and by disconnecting all nodes; hence forming an initial configuration satisfying all the requirements.

To prove the induction step, we distinguish two cases: depending on whether q_{i+1} was added as the target state of a broadcast action $!a$ from some $q \in S_i$; or whether q_{i+1} is the target state of a reception from some $q \in S_i$ with matching broadcast between two states already in S_i.

Case 1: There exists $q \in S_i$ with $q \xrightarrow{!a} q_{i+1}$. We apply the induction hypothesis to step i, and exhibit the various elements of the statement. Applying the copycat property for lossy broadcast systems (that is, Proposition 9) with node $\mathsf{n}_q^{\mathsf{main}}$, we build an execution $\rho' : \gamma'_0 \xrightarrow{*}_1 \gamma'$ such that $\gamma' = (\mathsf{N}, \mathsf{E}, \mathsf{L}')$ with $|\mathsf{N}'| = |\mathsf{N}| + 1$, and an appropriate injection ι. The fresh node $\mathsf{n}_{\mathsf{fresh}}$ is connected to nodes to which $\mathsf{n}_q^{\mathsf{main}}$ was connected before; hence, by induction hypothesis, it is only connected to nodes labelled with $\times$. Then we extend ρ' with $\gamma' \xrightarrow{\mathsf{n}_{\mathsf{fresh}}, !a} \gamma''$ and lose the message (this is for condition $\#\mathsf{nonlost\_steps}(\rho, \mathsf{n}_q^{\mathsf{main}}) = 0$ to be satisfied). We declare $\mathsf{n}_{q_{i+1}}^{\mathsf{main}} = \mathsf{n}_{\mathsf{fresh}}$. All requirements for γ'' are easily checked to be satisfied (when a node is labelled with $\times$ in γ', then it remains labelled by $\times$ in γ'').

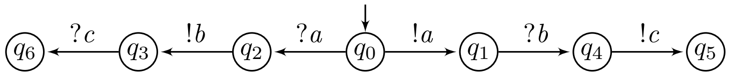


Figure 3 Illustrating example for the saturation algorithm.

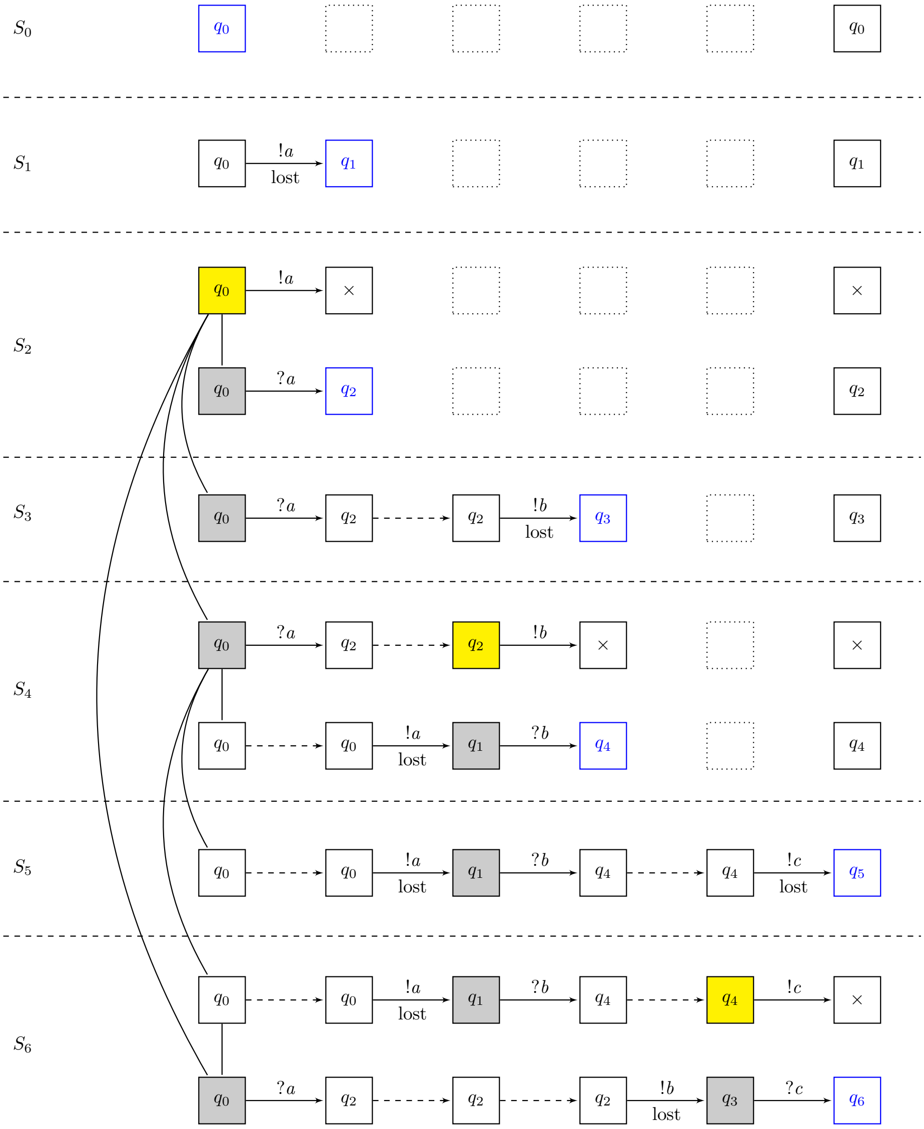


Figure 4 Applying saturation algorithm on protocol in Figure 3 in lossy semantics. Configurations are represented vertically; for readability, macrosteps merge several broadcasts.

Case 2: There exist $q, q', q'' \in S_i$ such that $q \xrightarrow{?a} q_{i+1}$ and $q' \xrightarrow{!a} q''$. We apply the induction hypothesis to step i, and exhibit the various elements of the statement. Applying twice the copycat property (that is, Proposition 9), once with node $\mathsf{n}_q^{\mathsf{main}}$ and once with node $\mathsf{n}_{q'}^{\mathsf{main}}$, we build an execution $\rho' : \gamma_0' \xrightarrow{*}_1 \gamma'$ such that $\gamma' = (\mathsf{N}, \mathsf{E}, \mathsf{L}')$ with $|\mathsf{N}'| = |\mathsf{N}| + 2$, and an appropriate injection ι. The two fresh nodes $\mathsf{n}_{\mathsf{fresh}}$ and $\mathsf{n}'_{\mathsf{fresh}}$ are only connected to ×-nodes in γ' (by induction hypothesis on $\mathsf{n}_q^{\mathsf{main}}$ and $\mathsf{n}_{q'}^{\mathsf{main}}$ respectively). We transform γ_0' into γ_0'' by connecting the two nodes $\mathsf{n}_{\mathsf{fresh}}$ and $\mathsf{n}'_{\mathsf{fresh}}$. By Proposition 9, we know that those two nodes don't perform any real sending (*i.e.*, $\#\mathsf{nonlost\_steps}(\rho', \mathsf{n}_{\mathsf{fresh}}) = 0$ and

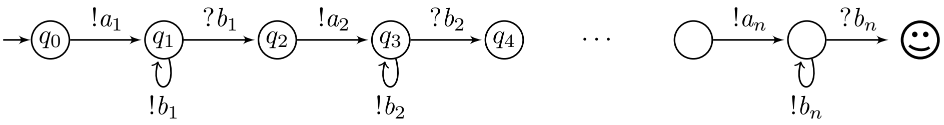


Figure 5 Broadcast protocol with linear cutoff and quadratic covering length.

$\#\mathsf{nonlost\_steps}(\rho', \mathsf{n}'_{\mathsf{fresh}}) = 0$), hence this new connection will not affect the labels of the nodes, and we can safely apply the same transitions as in ρ' from γ''_0 to get an execution $\rho'' : \gamma''_0 \xrightarrow{*}_1 \gamma''$, where γ'' coincides with γ', with an extra connection between nodes $\mathsf{n}_{\mathsf{fresh}}$ and $\mathsf{n}'_{\mathsf{fresh}}$. Then, we extend ρ'' with $\gamma'' \xrightarrow{\mathsf{n}'_{\mathsf{fresh}},!a} \gamma'''$. We assume it is a real sending, hence: node $\mathsf{n}_{\mathsf{fresh}}$ can progress from state q to q_{i+1}, and node $\mathsf{n}'_{\mathsf{fresh}}$ can progress from q' to q''. All other nodes which are connected to $\mathsf{n}'_{\mathsf{fresh}}$ are labelled by $\times$ in γ'', hence cannot be really affected by that sending. We relabel $\mathsf{n}'_{\mathsf{fresh}}$ to $\times$, and declare $\mathsf{n}^{\mathsf{main}}_{q_{i+1}} = \mathsf{n}_{\mathsf{fresh}}$. The expected conditions of the statement are easily checked to be satisfied by this new execution. ◀

Bounds are then obtained similarly to the reconfigurable case, see page 8.

3.3 Matching lower bounds for reconfigurable and lossy networks

In this section, we show that the linear bound on the cutoff and the quadratic bound on the length of witness executions are tight, both for the reconfigurable and the lossy broadcast networks.

▶ **Theorem 11.** *There exists a family of broadcast protocols $(\mathcal{P}_n)_n$ with $\mathcal{P}_n = (Q_n, I_n, \Sigma_n, \Delta_n)$, and target states $F_n \subseteq Q_n$ with $|Q_n| \in O(n)$, such that for every n, $\mathsf{COVER}_{\mathtt{r}}(\mathcal{P}_n, F_n) \neq \emptyset$, $\mathsf{COVER}_{\mathtt{l}}(\mathcal{P}_n, F_n) \neq \emptyset$, and any witness reconfigurable or lossy execution has size $O(n)$ and length $O(n^2)$.*

Proof. Consider $\mathcal{P}_n$, as depicted in Figure 5 with $2n{+}1$ states and $F_n = \{☺\}$. Any covering reconfigurable execution involves at least $n{+}1$ nodes, and has at least $\frac{n^2+5n}{2}$ steps. Indeed, intuitively, the process responsible for broadcasting b_i is blocked in q_{2i-1}, so that n such processes are needed, plus one process in ☺; moreover, $n{+}2{-}i$ broadcasts of a_i and one broadcast of each b_i happen. ◀

4 Succinctness of reconfigurations compared to losses

In this section, we show that reconfigurable executions can be linearly more succinct than lossy executions, in terms of number of nodes. Given the tight linear bound on cutoff, this is somehow optimal.

▶ **Theorem 12.** *There exists a family of broadcast protocols $(\mathcal{P}_n)_n$ with $\mathcal{P}_n = (Q_n, I_n, \Sigma_n, \Delta_n)$ and target states $F_n \subseteq Q_n$ such that for every n:*

- *there exists a reconfigurable covering execution in $\mathcal{P}_n$with 3 nodes; and*
- *any lossy covering execution in $\mathcal{P}_n$ requires $O(n)$ nodes.*

Proof. $\mathcal{P}_n$ is depicted in Figure 6. It has $3n{+}2$ states and we let $F_n = \{☺\}$. A covering reconfigurable execution of size 3 is given in Figure 7. Colored nodes broadcast a message in the step leading to the next configuration. Along that execution, the top node always remains at q_0 and alternatively broadcasts a to the middle node and disconnects; the middle node follows the chain of q_i states and alternatively broadcasts b_i's to the bottom node which gradually progresses along the chain of states r_i and reaches ☺.

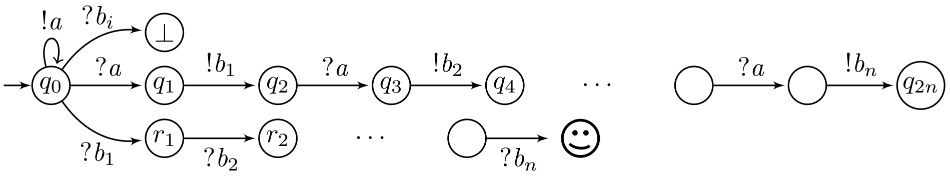


Figure 6 Example where reconfigurable semantics needs less nodes than lossy semantics.

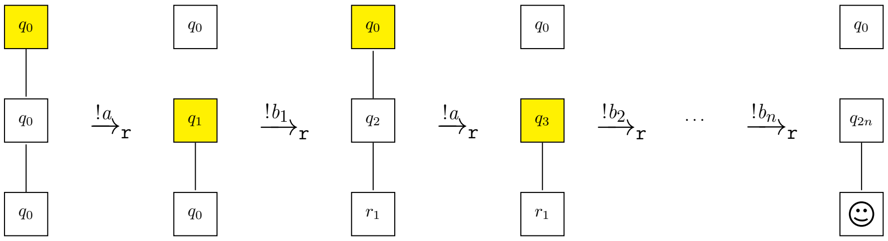


Figure 7 Covering reconfigurable execution with 3 nodes on the protocol from Figure 6.

Let us argue that in the lossy semantics, $O(n)$ nodes are needed to cover ☺. Obviously, one node, say $\mathsf{n}_{\smiley}$, is needed to reach the target state, after having received sequentially all the b_i's (which should then correspond to real broadcasts). Towards a contradiction, assume there is a node n which makes $\mathsf{n}_{\smiley}$ progress twice, that is, n is connected to $\mathsf{n}_{\smiley}$ and performs at least two real broadcasts, say $!b_i$ and $!b_j$ with $i < j$. Node n needs to receive $j - i > 0$ times the message a after the real $!b_i$ has occurred, hence there must be at least one node in state q_0 connected to n after the real $!b_i$ by n. This is not possible, since this node has received the real $!b_i$ while being in q_0, leading to $\perp$ if $i > 1$, otherwise $\perp$ or r_1. Hence, each broadcast $!b_i$ needs to be sent by a different node. This requires at least $n{+}1$ nodes, say $\{\mathsf{n}_i \mid 1 \leq i \leq n\} \cup \{\mathsf{n}_{\smiley}\}$: node n_i is responsible for broadcasting (with no loss) b_i and $\mathsf{n}_{\smiley}$ progresses towards ☺. Notice that $\mathsf{n}_{\smiley}$ might be the node responsible for broadcasting all the a's. We conclude that $n{+}1$ is a lower bound on the number of nodes needed to cover ☺ in the lossy semantics.

To complete this example, observe that $n{+}1$ nodes do actually suffice in lossy semantics to cover ☺. Let $\mathsf{N} = \{\mathsf{n}_i \mid 1 \leq i \leq n\} \cup \{\mathsf{n}_{\smiley}\}$ and consider the static communication topology defined by $\mathsf{n}_i \sim \mathsf{n}_{\smiley}$ for every i. In the covering lossy execution, node $\mathsf{n}_{\smiley}$ initially broadcasts a's, so that all its neighbours, the n_i's can move to q_{2i-1}, using lost sendings. Then the each node n_i broadcasts its message $b_i$to $\mathsf{n}_{\smiley}$, starting with n_1 until n_n, so that $\mathsf{n}_{\smiley}$ reaches ☺. ◀

5 Complexity of deciding the size of minimal witnesses

We now consider the following decision problem of determining the minimal size of coverability witnesses for both the reconfigurable and lossy semantics.

Minimum number of nodes for coverability (MinCover)
Input: A broadcast protocol $\mathcal{P}$, a set of states $F \subseteq Q$, and $k \in \mathbb{N}$.
Question: Does there exist a reconfigurable/lossy execution ρ covering some state in F, and with $\#\mathsf{nodes}(\rho) = k$?

By the copycat properties (for both semantics), if there is a covering execution of size less than k, then there is one of size exactly k.

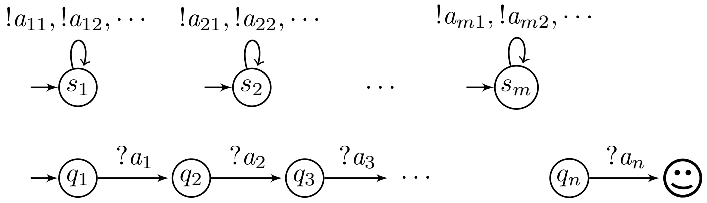

$!a_{11}, !a_{12}, \cdots$
$!a_{21}, !a_{22}, \cdots$
$!a_{m1}, !a_{m2}, \cdots$
s_1
s_2
$\cdots$
s_m
q_1
$?a_1$
q_2
$?a_2$
q_3
$?a_3$
$\cdots$
q_n
$?a_n$
☺

Figure 8 Illustration of the reduction to prove NP-hardness of MinCover.

▶ **Theorem 13.** MinCover *is* NP*-complete for both reconfigurable and lossy broadcast networks.*

The NP-hardness of MinCover is proved by reduction from SetCover, which is known to be NP-complete [9]. Recall that SetCover takes as input a finite set of elements $\mathcal{U}$, a collection $\mathcal{S}$ of subsets of $\mathcal{U}$ and an integer k, and returns yes iff there exists a subcollection of $\mathcal{S}$ of size at most k that covers $\mathcal{U}$.

Given an instance of the SetCover problem $(\mathcal{U}, \mathcal{S}, k)$ with $\mathcal{U} = \{a_1, a_2, \ldots, a_n\}$ and $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$, we build a protocol $\mathcal{P} = (Q, I, \Sigma, \Delta)$ as depicted in Figure 8, where we assume $S_i = \{a_{i1}, a_{i2}, \ldots\}$ for every i.

We can then show that $\mathcal{U}$ has a cover using $\mathcal{S}$ of size k if and only if there exists a reconfigurable/lossy execution for $\mathcal{P}$ covering F and with $k{+}1$ nodes.

For the NP-membership, it suffices to observe that the length of a minimal covering execution is polynomially bounded, thanks to Theorem 5 and 8. Moreover, configurations and updates of configurations by given transitions can be represented in and computed in a compact way. It is thus possible to implement a guess-and-check NP-algorithm for the MinCover problem, that non deterministically guesses an execution with k nodes of maximal length that is polynomially bounded in the size of the broadcast protocol.

6 Conclusion

In this paper, we have given a tight linear bound on the cutoff and a tight quadratic bound on the covering length for reconfigurable broadcast networks. We have also proposed a new semantics for broadcast networks with a static topology, where messages can be lost at sending. Similar tight bounds can be proven for that new semantics. Proofs are based on a refinement of the saturation algorithm of [2], and on fine analysis of copycat lemmas. As a side result of these constructions, we get that the set of states which can be covered by the two semantics is actually the same, but that the reconfigurable semantics can be linearly more succinct (in terms of number of nodes). We also prove the NP-completeness for the existence of a witness execution with the minimal number of nodes.

As future work, we want to pursue the study of the model with stochastic losses, and design analysis algorithms for various quantitative questions. Also, in this work we have not studied the tradeoff between number of nodes and length of covering computation. The precise interplay between number of nodes and length of covering execution is a possible direction for future work.

References

1 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. doi:10.2200/S00658ED1V01Y201508DCT013.

2 Giorgio Delzanno, Arnaud Sangnier, Riccardo Traverso, and Gianluigi Zavattaro. On the Complexity of Parameterized Reachability in Reconfigurable Broadcast Networks. In *Proceedings of the 32nd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'12)*, volume 18 of *Leibniz International Proceedings in Informatics*, pages 289–300. Leibniz-Zentrum für Informatik, December 2012. doi:10.4230/LIPIcs.FSTTCS.2012.289.

3 Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Parameterized Verification of Ad Hoc Networks. In *Proceedings of the 21st International Conference on Concurrency Theory (CONCUR'10)*, volume 6269 of *Lecture Notes in Computer Science*, pages 313–327. Springer, September 2010. doi:10.1007/978-3-642-15375-4_22.

4 Giorgio Delzanno, Arnaud Sangnier, and Gianluigi Zavattaro. Verification of Ad Hoc Networks with Node and Communication Failures. In *Proceedings of the 32nd International Conference on Formal Techniques for Distributed Systems (FMOODS/FORTE'12)*, volume 7273 of *Lecture Notes in Computer Science*. Springer, 2012.

5 E. Allen Emerson and A. Prasad Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1-2):105–131, August 1996. doi:10.1007/BF00625970.

6 Javier Esparza. Keeping a Crowd Safe: On the Complexity of Parameterized Verification (Invited Talk). In *Proceedings of the 31st Symposium on Theoretical Aspects of Computer Science (STACS'14)*, volume 25 of *Leibniz International Proceedings in Informatics*, pages 1–10. Leibniz-Zentrum für Informatik, March 2014. doi:10.4230/LIPIcs.STACS.2014.1.

7 Javier Esparza, Pierre Ganty, and Rupak Majumdar. Parameterized verification of asynchronous shared-memory systems. In *Proceedings of the 25th International Conference on Computer Aided Verification (CAV'13)*, volume 8044 of *Lecture Notes in Computer Science*, pages 124–140. Springer, July 2013. doi:10.1007/978-3-642-39799-8_8.

8 Steven M. German and A. Prasad Sistla. Reasoning about Systems with Many Processes. *Journal of the ACM*, 39(3):675–735, July 1992. doi:10.1145/146637.146681.

9 Richard M. Karp. Reducibility among Combinatorial Problems. In *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103, 1972.

10 Marvin Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall International, 1967.

11 Ichiro Suzuki. Proving Properties of a Ring of Finite-State Machines. *Information Processing Letters*, 28(4):213–214, 1988. doi:10.1016/0020-0190(88)90211-6.

Verification of Randomized Consensus Algorithms Under Round-Rigid Adversaries

Nathalie Bertrand
Univ. Rennes, Inria, CNRS, IRISA, Rennes, France
nathalie.bertrand@inria.fr

Igor Konnov
University of Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
igor.konnov@inria.fr

Marijana Lazić
TU München, 85748 Garching bei München, Germany
lazic@in.tum.de

Josef Widder
TU Wien, 1040 Vienna, Austria
Interchain Foundation, 6340 Baar, Switzerland
widder@forsyte.at

Abstract

Randomized fault-tolerant distributed algorithms pose a number of challenges for automated verification: (i) parameterization in the number of processes and faults, (ii) randomized choices and probabilistic properties, and (iii) an unbounded number of asynchronous rounds. This combination makes verification hard. Challenge (i) was recently addressed in the framework of threshold automata.

We extend threshold automata to model randomized consensus algorithms that perform an unbounded number of asynchronous rounds. For non-probabilistic properties, we show that it is necessary and sufficient to verify these properties under round-rigid schedules, that is, schedules where processes enter round r only after all processes finished round $r-1$. For almost-sure termination, we analyze these algorithms under round-rigid adversaries, that is, fair adversaries that only generate round-rigid schedules. This allows us to do compositional and inductive reasoning that reduces verification of the asynchronous multi-round algorithms to model checking of a one-round threshold automaton. We apply this framework and automatically verify the following classic algorithms: Ben-Or's and Bracha's seminal consensus algorithms for crashes and Byzantine faults, 2-set agreement for crash faults, and RS-Bosco for the Byzantine case.

2012 ACM Subject Classification Software and its engineering → Software verification; Software and its engineering → Software fault tolerance; Theory of computation → Logic and verification

Keywords and phrases threshold automata, counter systems, parameterized verification, randomized distributed algorithms, Byzantine faults

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.33

Related Version Full version with proofs is available at `https://hal.inria.fr/hal-01925533`.

Funding Supported by the Austrian Science Fund (FWF) via the National Research Network RiSE (S11403, S11405), project PRAVDA (P27722), and Doctoral College LogiCS (W1255-N23); by the Vienna Science and Technology Fund (WWTF) via project APALACHE (ICT15-103); and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 787367 (PaVeS). Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and others, see `https://www.grid5000.fr`.

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 33; pp. 33:1–33:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

```
1  bool v := input_value({0, 1});
2  int r := 1;
3  while (true) do
4    send (R,r,v) to all;
5    wait for n − t messages (R,r,∗);
6    if received (n + t) / 2 messages (R,r,w)
7    then send (P,r,w,D) to all;
8    else send (P,r,?) to all;
9    wait for n − t messages (P,r,∗);
10   if received at least t + 1
11       messages (P,r,w,D) then {
12     v := w;
13     if received at least (n + t) / 2
14       messages (P,r,w,D)
15     then decide w;
16   } else v := random({0, 1});
17   r := r + 1;
18 od
```

Figure 1 Pseudo code of Ben-Or's algorithm for Byzantine faults.

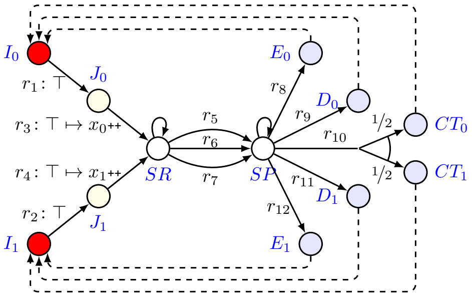
1 Introduction

Fault-tolerant distributed algorithms like Paxos and Blockchain recently receive much attention. Still, these systems are out of reach with current automated verification techniques. One problem comes from the scale: these systems should be verified for a very large (ideally even an unbounded) number of participants. In addition, many systems (including Blockchain), provide probabilistic guarantees. To check their correctness, one has to reason about randomized distributed algorithms in the parameterized setting.

In this paper, we make first steps towards parameterized verification of fault-tolerant randomized distributed algorithms. We consider consensus algorithms that follow the ideas of Ben-Or [3]. Interestingly, these algorithms were analyzed in [17, 15] where probabilistic reasoning was done using the probabilistic model checker PRISM [16] for systems of 10-20 processes, while only safety was verified in the parameterized setting using Cadence SMV. From a different perspective, these algorithms extend asynchronous threshold-guarded distributed algorithms from [12, 10] with two features (i) a random choice (coin toss), and (ii) repeated executions of the same algorithm until it converges (with probability 1).

A prominent example is Ben-Or's fault-tolerant consensus algorithm [3] given in Figure 1. It circumvents the impossibility of asynchronous consensus [9] by relaxing the termination requirement to almost-sure termination, *i.e.*, termination with probability 1. Here processes execute an infinite sequence of asynchronous loop iterations, which are called rounds r. Each round consists of two stages where they first exchange messages tagged R, wait until the number of received messages reaches a certain threshold (given as expression over parameters in line 5) and then exchange messages tagged P. In the code, n is the number of processes, among which at most t are Byzantine faulty (which may send conflicting information). The correctness of the algorithm should be verified for all values of the parameters n and t that meet a so-called resilience condition, *e.g.*, $n > 3t$. Carefully chosen thresholds ($n-t$, $(n+t)/2$ and $t+1$) on the number of received messages of a given type, ensure *agreement*, *i.e.*, that two correct processes never decide on different values. At the end of a round, if there is no "strong majority" for a value, *i.e.*, less than $(n+t)/2$ messages were received (cf. line 13), a process picks a new value randomly in line 16.

While these non-trivial threshold expressions can be dealt with using the methods in [10], several challenges remain. The technique in [10] can be used to verify one iteration of the round from Figure 1 only. However, consensus algorithms should prevent that there are no two rounds r and r' such that a process decides 0 in r and another decides 1 in r'. This calls for a compositional approach that allows one to compose verification results for individual rounds. A challenge in the composition is that distributed algorithms implement "asynchronous rounds", *i.e.*, during a run processes may be in different rounds at the same time.



$$r_5\colon x_0+x_1 \geq n-t-f \wedge x_0 \geq (n+t)/2-f \mapsto y_0\text{++}$$
$$r_6\colon x_0+x_1 \geq n-t-f \wedge x_1 \geq (n+t)/2-f \mapsto y_1\text{++}$$
$$r_7\colon x_0+x_1 \geq n-t-f \wedge x_0 \geq (n-3t)/2-f \\ \wedge\, x_1 \geq (n-3t)/2-f \mapsto y_?\text{++}$$
$$r_8\colon y_0+y_1+y_? \geq n-t-f \wedge y_? \geq (n-3t)/2-f \\ \wedge\, y_0 \geq t+1-f$$
$$r_9\colon y_0+y_1+y_? \geq n-t-f \wedge y_0 \geq (n+t)/2-f$$
$$r_{10}\colon y_0+y_1+y_? \geq n-t-f \wedge y_? \geq (n-3t)/2-f \\ \wedge\, y_? \geq n-2t-f-1$$

Figure 2 Ben-Or's algorithm as PTA with resilience condition $n > 3t \wedge t > 0 \wedge t \geq f \geq 0$.

In addition, the combination of distributed aspects and probabilities makes reasoning difficult. Quoting Lehmann and Rabin [18], "proofs of correctness for probabilistic distributed systems are extremely slippery". This advocates the development of automated verification techniques for probabilistic properties of randomized distributed algorithms in the parameterized setting.

Contributions. We extend the threshold automata framework from [10] to round-based algorithms with coin toss transitions. For the new framework we achieve the following:

1. For safety verification we introduce a method for compositional round-based reasoning. This allows us to invoke a reduction similar to the one in [8, 6, 7]. We highlight necessary fairness conditions on individual rounds. This provides us with specifications to be checked on a one-round automaton.
2. We reduce probabilistic liveness verification to proving termination with positive probability within a fixed number of rounds. To do so, we restrict ourselves to round-rigid adversaries, that is, adversaries that respect the round ordering. In contrast to existing work that proves almost-sure termination for fixed number of participants, these are the first parameterized model checking results for probabilistic properties.
3. We check the specifications that emerge from points 1. and 2. and thus verify challenging benchmarks in the parameterized setting. We verify Ben-Or's [3] and Bracha's [5] classic algorithms, and more recent algorithms such as 2-set agreement [21], and RS-Bosco [23].

2 Overview

We introduce probabilistic threshold automata to model randomized threshold-based algorithms. An example of such an automaton is given in Figure 2. Nodes represent local states (or locations) of processes, which move along the labeled edges or forks. Edges and forks are called rules. Labels have the form $\varphi \mapsto u$, meaning that a process can move along the edge only if φ evaluates to true, and this is followed by the update u of shared variables. Additionally, each tine of a fork is labeled with a number in the $[0, 1]$ interval, representing the probability of a process moving along the fork to end up at the target location of the tine. If we ignore the dashed arrows in Figure 2, a threshold automaton captures the behavior of a process in one round, that is, a loop iteration in Figure 1.

The code in Figure 1 refers to numbers of received messages and, as is typical for distributed algorithms, their relation to sent messages (that is the semantics of send and receive) is not explicit in the pseudo code. To formalize the behavior, the encoding in the threshold automaton directly refers to the numbers of sent messages, and they are encoded in the shared variables x_i and y_i. The algorithm is parameterized: n is the number of processes, t is the assumed number of faults and f is the actual number of faults. It should be demonstrated to work under the resilience condition $n > 3t \wedge t \geq f \wedge t > 0$. For instance, the locations J_0 and J_1 capture that a loop is entered with v being 0 and 1, respectively. Sending an $(R, r, 0)$ and $(R, r, 1)$ message is captured by the increments on the shared variables x_0 and x_1 in the rules r_3 and r_4, respectively; e.g., a process that is in location J_0 uses rule r_3 to go to location SR ("sent R message"), and increments x_0 in doing so. Waiting for R and P messages in the lines 5 and 9, is captured by looping in the locations SR and SP. In line 7 a process sends, e.g., a $(P, r, 0, D)$ message if it has *received* $n - t$ messages out of which $(n + t)/2$ are $(R, r, 0)$ messages. This is captured in the guard of rule r_5 where $x_0 + x_1 \geq n - t - f$ checks the number of received messages in total, and $x_0 \geq (n + t)/2 - f$ checks for the specific messages containing 0. The "$-f$" term models that in the message passing semantics underlying Figure 1, f messages from Byzantine faults may be received *in addition* to the messages sent by correct processes (modeled by shared variables in Figure 2). The branching at the end of the loop from lines 10 to 18 is captured by the rules outgoing of SP. In particular rule r_{10} captures the coin toss in line 16. The non-determinism due to faults and asynchrony is captured by multiple rules being enabled in the same configuration.

Liveness properties of distributed algorithms typically require fairness constraints, e.g., every message sent by a correct process to a correct process is eventually received. For instance, this implies in Figure 1 that if $n - t$ correct processes have sent messages of the form $(R, 1, *)$ and $(n + t)/2$ correct processes have sent messages of the form $(R, 1, 0)$ then every correct process should eventually execute line 7, and proceed to line 9. We capture this by the following fairness constraint: if $x_0 + x_1 \geq n - t \wedge x_0 \geq (n + t)/2-$ that is, rule r_5 is enabled without the help of the f faulty processes but by "correct processes alone"$-$ then the source location of rule r_5, namely SR should eventually be evacuated, that is, its corresponding counter should eventually be 0.

The dashed edges, called round switch rules, encode how a process, after finishing a round, starts the next one. The round number r serves as the loop iterator in Figure 1, and in each iteration, processes send messages that carry r. To capture this, our semantics will introduce fresh shared variables initialized with 0 for each round r. Because there are infinitely many rounds, this means a priori we have infinitely many variables.

As parameterized verification of threshold automata is in general undecidable [14], we consider the so-called "canonic" restrictions here, *i.e.*, only increments on shared variables, and no increments of the same variable within loops. These restrictions still allow us to model many threshold-based fault-tolerant distributed algorithms [10]. As a result, threshold automata without probabilistic forks and round switching rules can be automatically checked for safety and liveness [10]. Adding forks and round switches is required to adequately model randomized distributed algorithms. Here we will use a convenient restriction that requires that coin-toss transitions only appear at the end of a round, e.g., line 16 of Figure 1. Intuitively, as discussed in Section 1, a coin-toss is only necessary if there is no strong majority. Thus, all our benchmarks have this feature, and we exploit it in Section 7.

In order to overcome the issue of infinitely many rounds, we prove in Section 6 that we can verify probabilistic threshold automata by analyzing a one-round automaton that fits in the framework of [10]. We prove that we can reorder transitions of any fair execution such that their round numbers are in an increasing order. The obtained ordered execution is stutter equivalent with the original one, and thus, they satisfy the same $\mathsf{LTL}_{\text{-}\mathsf{X}}$ properties over the atomic propositions describing only one round. In other words, our targeted concurrent systems can be transformed to a sequential composition of one-round systems.

The main problem with isolating a one-round system is that consensus specifications often talk about at least two different rounds. In this case we need to use round invariants that imply the specifications. For example, if we want to verify agreement, we have to check whether two processes decide different values, possibly in different rounds. We do this in two steps: (i) we check the round invariant that no process changes its decision from round to round, and (ii) we check that within a round no two processes disagree.

Finally, verifying almost-sure termination under round-rigid adversaries calls for distinct arguments. Our methodology follows the lines of the manual proof of Ben Or's consensus algorithm by Aguilera and Toueg [1]. However, our arguments are not specific to Ben Or's algorithm, and we apply it to other randomized distributed algorithms (see Section 8). Compared to their paper-and-pencil proof, the threshold automata framework required us to provide a more formal setting and a more informative proof, also pinpointing the needed hypothesis. The crucial parts of our proof are automatically checked by the model checker ByMC [13]. Hence the established correctness stands on less slippery ground, which addresses the mentioned concerns of Lehmann and Rabin.

3 The Probabilistic Threshold Automata Framework

A *probabilistic threshold automaton* PTA is a tuple $(\mathcal{L}, \mathcal{V}, \mathcal{R}, RC)$, where

- $\mathcal{L}$ is a finite set of locations, that contains the following disjoint subsets: *initial locations* $\mathcal{I}$, *final locations* $\mathcal{F}$, and *border locations* $\mathcal{B}$, with $|\mathcal{B}| = |\mathcal{I}|$;
- $\mathcal{V}$ is a set of variables. It is partitioned in two sets: Π contains *parameter variables*, and Γ contains *shared variables*;
- $\mathcal{R}$ is a finite set of *rules*; and
- RC, the *resilience condition*, is a formula in linear integer arithmetic over parameter variables.

In the following we introduce rules in detail, and give syntactic restrictions on locations. The resilience condition RC only appears in the definition of the semantics in Section 3.1.

A rule r is a tuple $(\mathit{from}, \delta_{to}, \varphi, \vec{u})$ where $\mathit{from} \in \mathcal{L}$ is the *source* location, $\delta_{to} \in \mathsf{Dist}(\mathcal{L})$ is a probability distribution over the *destination* locations, $\vec{u} \in \mathbb{N}_0^{|\Gamma|}$ is the *update vector*, and φ is a guard, *i.e.*, a conjunction of expressions of the form $b \cdot x \geq \bar{a} \cdot \mathbf{p}^\intercal + a_0$ or $b \cdot x < \bar{a} \cdot \mathbf{p}^\intercal + a_0$, where $x \in \Gamma$ is a shared variable, $\bar{a} \in \mathbb{Z}^{|\Pi|}$ is a vector of integers, $a_0, b \in \mathbb{Z}$, and $\mathbf{p}$ is the vector of all parameters. If $r.\delta_{to}$ is a Dirac distribution, *i.e.*, there exists $\ell \in \mathcal{L}$ such that $r.\delta_{to}(\ell) = 1$, we call r a *Dirac rule*, and write it as $(\mathit{from}, \ell, \varphi, \vec{u})$. Destination locations of non-Dirac rules are in $\mathcal{F}$ (coin-toss transitions only happen at the end of a round).

Probabilistic threshold automata model algorithms with successive identical rounds. Informally, a round happens between border locations and final locations. Then round switch rules let processes move from final locations of a given round to border locations of the next round. From each border location there is exactly one Dirac rule to an initial location, and it has a form $(\ell, \ell', \mathtt{true}, \vec{0})$ where $\ell \in \mathcal{B}$ and $\ell' \in \mathcal{I}$. As $|\mathcal{B}| = |\mathcal{I}|$, one can think of border locations as copies of initial locations. It remains to model from which final locations to which border location (that is, initial for the next round) processes move. This is done by *round switch rules*. They can be described as Dirac rules $(\ell, \ell', \mathtt{true}, \vec{0})$ with $\ell \in \mathcal{F}$ and $\ell' \in \mathcal{B}$. The set of round switch rules is denoted by $\mathcal{S} \subseteq \mathcal{R}$.

A location is in $\mathcal{B}$ if and only if all the incoming edges are in $\mathcal{S}$. Similarly, a location is in $\mathcal{F}$ if and only if there is only one outgoing edge and it is in $\mathcal{S}$.

Figure 2 depicts a PTA with border locations $\mathcal{B} = \{I_0, I_1\}$, initial locations $\mathcal{I} = \{J_0, J_1\}$, and final locations $\mathcal{F} = \{E_0, E_1, D_0, D_1, CT_0, CT_1\}$. The only rule that is not Dirac rule is r_{10}, and round switch rules are represented by dashed arrows.

3.1 Probabilistic Counter Systems

A resilience condition RC defines the set of *admissible parameters* $\mathbf{P}_{RC} = \{\mathbf{p} \in \mathbb{N}_0^{|\Pi|} : \mathbf{p} \models RC\}$. We introduce a function $N\colon \mathbf{P}_{RC} \to \mathbb{N}_0$ that maps a vector of admissible parameters to a number of modeled processes in the system. For instance, for the automaton in Figure 2, N is the function $(n,t,f) \mapsto n-f$, as we model only the $n-f$ correct processes explicitly, while the effect of faulty processes is captured in non-deterministic choices between different guards as discussed in Section 2. Given a PTA and a function N, we define the semantics, called *probabilistic counter system* Sys(PTA), to be the infinite-state MDP $(\Sigma, I, \mathsf{Act}, \Delta)$, where Σ is the set of configurations for PTA among which $I \subseteq \Sigma$ are initial, the set of actions is $\mathsf{Act} = \mathcal{R} \times \mathbb{N}_0$ and $\Delta\colon \Sigma \times \mathsf{Act} \to \mathsf{Dist}(\Sigma)$ is the probabilistic transition function.

Configurations. In a configuration $\sigma = (\vec{\boldsymbol{\kappa}}, \vec{g}, \mathbf{p})$, a function $\sigma.\vec{\boldsymbol{\kappa}}\colon \mathcal{L} \times \mathbb{N}_0 \to \mathbb{N}_0$ defines values of location counters per round, a function $\sigma.\vec{g}\colon \Gamma \times \mathbb{N}_0 \to \mathbb{N}_0$ defines shared variable values per round, and a vector $\sigma.\mathbf{p} \in \mathbb{N}_0^{|\Pi|}$ defines parameter values. We denote the vector $(\vec{g}[x,k])_{x\in\Gamma}$ of shared variables in a round k by $\vec{g}[k]$, and by $\vec{\boldsymbol{\kappa}}[k]$ we denote the vector $(\vec{\boldsymbol{\kappa}}[\ell,k])_{\ell\in\mathcal{L}}$ of local state counters in a round k.

A configuration $\sigma = (\vec{\boldsymbol{\kappa}}, \vec{g}, \mathbf{p})$ is *initial* if all processes are in initial states of round 0, and all global variables evaluate to 0, that is, if for every $x \in \Gamma$ and $k \in \mathbb{N}_0$ we have $\sigma.\vec{g}[x,k] = 0$, if $\sum_{\ell\in\mathcal{B}} \sigma.\vec{\boldsymbol{\kappa}}[\ell,0] = N(\mathbf{p})$, and finally if $\sigma.\vec{\boldsymbol{\kappa}}[\ell,k] = 0$, for every $(\ell,k) \in (\mathcal{L}\setminus\mathcal{B}) \times \{0\} \cup \mathcal{L} \times \mathbb{N}$.

A threshold guard evaluates to true in a configuration σ for a round k, written $\sigma, k \models \varphi$, if for all its conjuncts $b \cdot x \geq \bar{a} \cdot \mathbf{p}^\intercal + a_0$, it holds that $b \cdot \sigma.\vec{g}[x,k] \geq \bar{a} \cdot (\sigma.\mathbf{p}^\intercal) + a_0$ (and similarly for conjuncts of the other form, *i.e.*, $b \cdot x < \bar{a} \cdot \mathbf{p}^\intercal + a_0$).

Actions. An action $\alpha = (r,k) \in \mathsf{Act}$ stands for the execution of a rule r in round k (by a single process). We write $\alpha.\mathit{from}$ for $r.\mathit{from}$, $\alpha.\delta_{to}$ for $r.\delta_{to}$, etc. An action $\alpha = (r,k)$ is *unlocked* in a configuration σ, if its guard evaluates to true in its round, that is $\sigma, k \models r.\varphi$. An action $\alpha = (r,k)$ is *applicable* to a configuration σ if α is unlocked in σ, and there is at least one process in the source location $r.\mathit{from}$, formally, $\sigma.\vec{\boldsymbol{\kappa}}[r.\mathit{from},k] \geq 1$. When an action α is applicable to σ, and when ℓ is a potential destination location for the probabilistic action α, we write $\mathit{apply}(\sigma,\alpha,\ell)$ for the resulting configuration: parameters are unchanged, shared variables are updated according to the update vector $r.\vec{u}$, and the values of counters are modified in a natural way: as a process moves from $r.\mathit{from}$ to ℓ in round k, counter $\vec{\boldsymbol{\kappa}}[r.\mathit{from},k]$ is decreased by 1 and $\vec{\boldsymbol{\kappa}}[\ell,k]$ is increased by 1. The probabilistic transition function Δ is defined by: $\Delta(\sigma,\alpha)(\sigma') = \alpha.\delta_{to}(\ell)$ if $\mathit{apply}(\sigma,\alpha,\ell) = \sigma'$, and 0 otherwise.

3.2 Non-probabilistic Counter Systems

Non-probabilistic threshold automata are defined in [12], and they can be seen as a special case of probabilistic threshold automata where all rules are Dirac rules.

With a PTA, one can naturally associate a non-probabilistic threshold automaton, by replacing probabilities with non-determinism.

▶ **Definition 1.** *Given a PTA* $= (\mathcal{L}, \mathcal{V}, \mathcal{R}, RC)$*,* its (non-probabilistic) threshold automaton *is* $\mathsf{TA}_{\mathsf{PTA}} = (\mathcal{L}, \mathcal{V}, \mathcal{R}_{np}, RC)$ *where the set of rules* $\mathcal{R}_{np}$ *is defined as* $\{r_\ell = (\mathit{from}, \ell, \varphi, \vec{u}) : r = (\mathit{from}, \delta_{to}, \varphi, \vec{u}) \in \mathcal{R} \wedge \ell \in \mathcal{L} \wedge \delta_{to}(\ell) > 0\}$*.*

We write TA instead of $\mathsf{TA}_{\mathsf{PTA}}$ when the automaton PTA is clear from the context. Every rule from $\mathcal{R}_{np}$ corresponds to exactly one rule in $\mathcal{R}$, and for every rule in $\mathcal{R}$ there is at least one corresponding rule in $\mathcal{R}_{np}$ (and exactly one for Dirac rules).

If we understand a TA as a PTA where all rules are Dirac rules, we can define transitions using the partial function *apply* in order to obtain an infinite (non-probabilistic) counter system, which we denote by $\mathsf{Sys}_\infty(\mathsf{TA})$. Moreover, since in this case $\mathcal{R} = \mathcal{R}_{np}$, actions of the PTA exactly match transitions of its TA. We obtain σ' by applying $t = (r, k)$ to σ, and write this as $\sigma' = t(\sigma)$, if and only if for the destination location ℓ of r holds that $apply(\sigma, t, \ell) = \sigma'$.

Also, starting from a PTA, one can define the counter system $\mathsf{Sys}(\mathsf{PTA})$, and consequently its non-probabilistic counterpart $\mathsf{Sys}_{np}(\mathsf{PTA})$. As the definitions of $\mathsf{Sys}_{np}(\mathsf{PTA})$ and $\mathsf{Sys}_\infty(\mathsf{TA})$ are equivalent for a given PTA, we are free to choose one, and always use $\mathsf{Sys}_\infty(\mathsf{TA})$.

A (finite or infinite) sequence of transitions is called *schedule*, and it is often denoted by τ. A schedule $\tau = t_1, t_2, \dots, t_{|\tau|}$ is applicable to a configuration σ if there is a sequence of configurations $\sigma = \sigma_0, \sigma_1, \dots, \sigma_{|\tau|}$ such that for every $1 \leq i \leq |\tau|$ we have that t_i is applicable to σ_{i-1} and $\sigma_i = t_i(\sigma_{i-1})$. A *path* is an alternating sequence of configurations and transitions, for example $\sigma_0, t_1, \sigma_1, \dots, t_{|\tau|}, \sigma_{|\tau|}$, such that for every t_i, $1 \leq i \leq |\tau|$, in the sequence, we have that t_i is applicable to σ_{i-1} and $\sigma_i = t_i(\sigma_{i-1})$. Given a configuration σ_0 and a schedule $\tau = t_1, t_2, \dots, t_{|\tau|}$, we denote by $\mathsf{path}(\sigma_0, \tau)$ a path $\sigma_0, t_1, \sigma_1, \dots, t_{|\tau|}, \sigma_{|\tau|}$ where $t_i(\sigma_{i-1}) = \sigma_i$, $1 \leq i \leq |\tau|$. Similarly we define an infinite schedule $\tau = t_1, t_2, \dots$, and an infinite path $\sigma_0, t_1, \sigma_1, \dots$, also denoted by $\mathsf{path}(\sigma_0, \tau)$. An infinite path is *fair* if no transition is applicable forever from some point on. Equivalently, when a transition is applicable, eventually either its guard becomes false, or all processes leave its source location.

Since every transition in $\mathsf{Sys}_\infty(\mathsf{TA})$ comes from an action in $\mathsf{Sys}(\mathsf{PTA})$, note that every path in $\mathsf{Sys}_\infty(\mathsf{TA})$ is a valid path in $\mathsf{Sys}(\mathsf{PTA})$.

3.3 Adversaries

As usual, the non-determinism is resolved by a so-called adversary. Let Paths be the set of all finite paths in $\mathsf{Sys}(\mathsf{PTA})$. An *adversary* is a function $\mathsf{a} : \mathsf{Paths} \to \mathsf{Act}$, that given a finite path π selects an action applicable to the last configuration of π. Given a configuration σ and an adversary a, we generate a family of paths, depending on the outcomes of non-Dirac transitions. We denote this set by $\mathsf{paths}(\sigma, \mathsf{a})$. An adversary a is *fair* if all paths in $\mathsf{paths}(\sigma, \mathsf{a})$ are fair. As usual, the Markov Decision Process (MDP) $\mathsf{Sys}(\mathsf{PTA})$ together with an initial configuration σ and an adversary a induce a Markov chain, written $\mathcal{M}^\sigma_\mathsf{a}$. We write $\mathbb{P}^\sigma_\mathsf{a}$ for the probability measure over infinite paths starting at σ in the latter Markov chain.

We call an adversary a *round-rigid* if it is fair, and if every sequence of actions it produces can be decomposed to a concatenation of sequences of action of the form $s_1 \cdot s_1^p \cdot s_2 \cdot s_2^p \ldots$, where for all $k \in \mathbb{N}$, we have that s_k contains only Dirac actions of round k, and s_k^p contains only non-Dirac actions of round k. We denote the set of all round-rigid adversaries by $\mathcal{A}^{\mathrm{R}}$.

3.4 Atomic Propositions and Stutter Equivalence

The atomic propositions we consider describe non-emptiness of a location $\ell \in \mathcal{L} \setminus \mathcal{B}$ in a round k, *i.e.*, whether there is at least one process in location ℓ in round k. Formally, the set of all such propositions for a round $k \in \mathbb{N}_0$ is denoted by $\mathrm{AP}_k = \{\mathrm{p}(\ell, k) \colon \ell \in \mathcal{L} \setminus \mathcal{B}\}$. For every k we define a labeling function $\lambda_k \colon \Sigma \to 2^{\mathrm{AP}_k}$ such that $\mathrm{p}(\ell, k) \in \lambda_k(\sigma)$ iff $\sigma.\vec{\boldsymbol{\kappa}}[\ell, k] > 0$. By abusing notation, we write "$\vec{\boldsymbol{\kappa}}[\ell, k] > 0$" and "$\vec{\boldsymbol{\kappa}}[\ell, k] = 0$" instead of $\mathrm{p}(\ell, k)$ and $\neg\mathrm{p}(\ell, k)$, resp.

We denote by $\pi_1 \triangleq_k \pi_2$ that the paths π_1 and π_2 are stutter equivalent [2] w.r.t. AP_k. Two counter systems C_0 and C_1 are stutter equivalent w.r.t. AP_k, written $C_0 \triangleq_k C_1$, if for every path π from C_i there is a path π' from C_{1-i} such that $\pi \triangleq_k \pi'$, for $i \in \{0, 1\}$.

4 Consensus Properties and their Verification

In probabilistic (binary) consensus every correct process has an initial value from $\{0,1\}$. It consists of safety specifications and an almost-sure termination requirement, which we consider in its round-rigid variant:

Agreement: No two correct processes decide differently.

Validity: If all correct processes have v as the initial value, then no process decides $1-v$.

Probabilistic wait-free termination: Under every round-rigid adversary, with probability 1 every correct process eventually decides.

We now discuss the formalization of these specifications in the context of Ben-Or's algorithm whose threshold automaton is given in Figure 2.

Formalization. In order to formulate and analyze the specifications, we partition every set $\mathcal{I}$, $\mathcal{B}$, and $\mathcal{F}$, into two subsets $\mathcal{I}_0 \uplus \mathcal{I}_1$, $\mathcal{B}_0 \uplus \mathcal{B}_1$, and $\mathcal{F}_0 \uplus \mathcal{F}_1$, respectively. For every $v \in \{0,1\}$, the partitions satisfy the following:

(R1) The processes that are initially in a location $\ell \in \mathcal{I}_v$ have the initial value v.

(R2) Rules connecting locations from $\mathcal{B}$ and $\mathcal{I}$ respect the partitioning, *i.e.*, they connect $\mathcal{B}_v$ and $\mathcal{I}_v$. Similarly, rules connecting locations from $\mathcal{F}$ and $\mathcal{B}$ respect the partitioning.

We introduce two subsets $\mathcal{D}_v \subseteq \mathcal{F}_v$, for $v \in \{0,1\}$. Intuitively, a process is in $\mathcal{D}_v$ in a round k if and only if it decides v in that round. Now we can express the specifications as follows:

Agreement: For both values $v \in \{0,1\}$ the following holds:

$$\forall k \in \mathbb{N}_0, \forall k' \in \mathbb{N}_0.\ \mathsf{A}\left(\mathsf{F} \bigvee\nolimits_{\ell \in \mathcal{D}_v} \vec{\boldsymbol{\kappa}}[\ell,k] > 0 \ \rightarrow\ \mathsf{G} \bigwedge\nolimits_{\ell' \in \mathcal{D}_{1-v}} \vec{\boldsymbol{\kappa}}[\ell',k'] = 0\right) \tag{1}$$

Validity: For both $v \in \{0,1\}$ it holds

$$\forall k \in \mathbb{N}_0.\ \mathsf{A}\left(\bigwedge\nolimits_{\ell \in \mathcal{I}_v} \vec{\boldsymbol{\kappa}}[\ell,0] = 0 \ \rightarrow\ \mathsf{G} \bigwedge\nolimits_{\ell' \in \mathcal{D}_v} \vec{\boldsymbol{\kappa}}[\ell',k] = 0\right) \tag{2}$$

Probabilistic wait-free termination: For every round-rigid adversary a

$$\mathbb{P}_{\mathsf{a}}\left(\bigvee\nolimits_{k \in \mathbb{N}_0} \bigvee\nolimits_{v \in \{0,1\}} \mathsf{G} \bigwedge\nolimits_{\ell \in \mathcal{F} \setminus \mathcal{D}_v} \vec{\boldsymbol{\kappa}}[\ell,k] = 0\right) \quad = \quad 1 \tag{3}$$

Agreement and validity are non-probabilistic properties, and thus can be analyzed on the non-probabilistic counter system $\mathsf{Sys}_\infty(\mathsf{TA})$. For verifying probabilistic wait-free termination, we make explicit the following assumption that is present in all our benchmarks: all non-Dirac transitions have non-zero probability to lead to an $\mathcal{F}_v$ location, for both values $v \in \{0,1\}$.

In Section 5 we formalize safety specifications and reduce them to single-round specifications. In Section 6 we reduce verification of multi-round counter systems to verification of single-round systems. In Section 7 we discuss our approach to probabilistic termination.

5 Reduction to Specifications with one Round Quantifier

Agreement contains two round variables k and k', and Validity considers rounds 0 and k. Thus, both involve two round numbers. As ByMC can only analyze one round systems [10], the properties are only allowed to use one round number. In this section we show how to check formulas (1) and (2) by checking properties that refer to one round. Namely, we introduce round invariants (4) and (5), and prove that they imply Agreement and Validity.

The first round invariant claims that in every round and in every path, once a process decides v in a round, no process ever enters a location from $\mathcal{F}_{1-v}$ in that round. Formally:

$$\forall k \in \mathbb{N}_0.\ \mathsf{A}\left(\mathsf{F} \bigvee\nolimits_{\ell \in \mathcal{D}_v} \vec{\boldsymbol{\kappa}}[\ell, k] > 0 \quad \rightarrow \quad \mathsf{G} \bigwedge\nolimits_{\ell' \in \mathcal{F}_{1-v}} \vec{\boldsymbol{\kappa}}[\ell', k] = 0\right) \tag{4}$$

The second round invariant claims that in every round in every path, if no process starts a round with a value v, then no process terminates that round with value v. Formally:

$$\forall k \in \mathbb{N}_0.\ \mathsf{A}\left(\mathsf{G} \bigwedge\nolimits_{\ell \in \mathcal{I}_v} \vec{\boldsymbol{\kappa}}[\ell, k] = 0 \quad \rightarrow \quad \mathsf{G} \bigwedge\nolimits_{\ell' \in \mathcal{F}_v} \vec{\boldsymbol{\kappa}}[\ell', k] = 0\right) \tag{5}$$

The following proposition is proved using restriction (R2) and by an inductive argument over the round number.

▶ **Proposition 2.** *If $\mathsf{Sys}_\infty(\mathsf{TA}) \models (4) \wedge (5)$, then $\mathsf{Sys}_\infty(\mathsf{TA}) \models (1) \wedge (2)$.*

6 Reduction to Single-Round Counter System

Given a property of one round, our goal is to prove that there is a counterexample to the property in the multi-round system iff there is a counterexample in a single-round system. This is formulated in Theorem 6, and it allows us to use ByMC on a single-round system.

The proof idea contains two parts. First, in Section 6.1 we prove that one can replace an arbitrary finite schedule with a round-rigid one, while preserving atomic propositions of a fixed round. We show that swapping two adjacent transitions that do not respect the order over round numbers in an execution, gives us a legal stutter equivalent execution, *i.e.*, an execution satisfying the same $\mathsf{LTL}_{\text{-}\mathsf{X}}$ properties.

Second, in Section 6.2 we extend this reasoning to infinite schedules, and lift it from schedules to transition systems. The main idea is to do inductive and compositional reasoning over the rounds. To do so, we need well-defined round boundaries, which is the case if every round that is started is also finished; a property we can automatically check for fair schedules. In more detail, regarding propositions for one round, we show that the multi-round transition system is stutter equivalent to a single-round transition system. This holds under the assumption that all fair executions of a single-round transition system terminate, and this can be checked using the technique from [10]. As stutter equivalence of systems implies preservation of $\mathsf{LTL}_{\text{-}\mathsf{X}}$ properties, this is sufficient to prove the main goal of the section.

6.1 Reduction from arbitrary schedules to round-rigid schedules

▶ **Definition 3.** *A schedule $\tau = (r_1, k_1) \cdot (r_2, k_2) \cdot \ldots \cdot (r_m, k_m)$, $m \in \mathbb{N}_0$, is called* round-rigid *if for every $1 \le i < j \le m$, we have $k_i \le k_j$.*

The following proposition shows that any finite schedule can be re-ordered into a round-rigid one that is stutter equivalent regarding $\mathsf{LTL}_{\text{-}\mathsf{X}}$ formulas over proposition from AP_k, for all rounds k. It is proved using arguments on the commutativity of transitions, similar to [8].

▶ **Proposition 4.** *For every configuration σ and every finite schedule τ applicable to σ, there is a round-rigid schedule τ' such that the following holds:*

(a) *Schedule τ' is applicable to configuration σ.*
(b) *τ' and τ reach the same configuration when applied to σ,* i.e., *$\tau'(\sigma) = \tau(\sigma)$.*
(c) *For every $k \in \mathbb{N}_0$ we have $\mathit{path}(\sigma, \tau) \triangleq_k \mathit{path}(\sigma, \tau')$.*

Thus, instead of reasoning about all finite schedules of $\mathsf{Sys}_\infty(\mathsf{TA})$, it is sufficient to reason about its round-rigid schedules. In the following section we use this to simplify the verification further, namely to a single-round counter system.

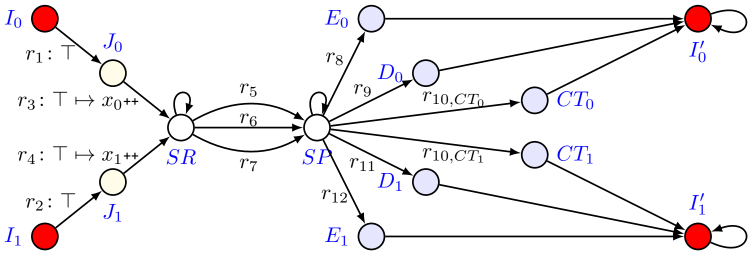


Figure 3 The single-round threshold automaton $\mathsf{TA}^{\mathrm{rd}}$ obtained from PTA in Figure 2.

6.2 From round-rigid schedules to single-round counter system

For each PTA, we define a *single-round threshold automaton* that can be analyzed with the tools of [12] and [10]. Roughly speaking, we focus on one round, but also keep the border locations of the next round, where we add self-loops. Figure 3 represents the single-round threshold automaton associated with the PTA from Figure 2. We can prove that for specific fairness constraints, this automaton shows the same behavior as a round in $\mathsf{Sys}_\infty(\mathsf{TA})$.

We restrict ourselves to fair schedules, that is, those where no transition is applicable forever. We also assume that every fair schedule of a single-round system terminates, *i.e.*, eventually every process reaches a state from $\mathcal{B}'$. Under the fairness assumption we check the latter assumption with ByMC [13]. Moreover, we restrict ourselves to non-blocking threshold automata, that is, we require that in each configuration each location has at least one outgoing rule unlocked. As we use TAs to model distributed algorithms, this is no restriction: locations in which no progress should be made unless certain thresholds are reached, typically have self-loops that are guarded with $\texttt{true}$ (e.g. SR and SP). Thus for our benchmarks one can easily check whether they are non-blocking using SMT (we have to check that there is no evaluation of the variables such that all outgoing rules are disabled).

▶ **Definition 5.** *Given a* $\mathsf{PTA} = (\mathcal{L}, \mathcal{V}, \mathcal{R}, RC)$ *or its* $\mathsf{TA} = (\mathcal{L}, \mathcal{V}, \mathcal{R}_{np}, RC)$, *we define a* single-round threshold automaton $\mathsf{TA}^{rd} = (\mathcal{L} \cup \mathcal{B}', \mathcal{V}, \mathcal{R}^{rd}, RC)$, *where* $\mathcal{B}' = \{\ell' : \ell \in \mathcal{B}\}$ *are copies of border locations, and* $\mathcal{R}^{rd} = (\mathcal{R}_{np} \setminus \mathcal{S}) \cup \mathcal{S}' \cup \mathcal{R}^{loop}$, *where* $\mathcal{R}^{loop} = \{(\ell', \ell', \texttt{true}, \vec{0}) : \ell' \in \mathcal{B}'\}$ *are self-loop rules at locations from* $\mathcal{B}'$ *and* $\mathcal{S}' = \{(from, \ell', \texttt{true}, \vec{0}) : (from, \ell, \texttt{true}, \vec{0}) \in \mathcal{S}$ *with* $\ell' \in \mathcal{B}'\}$ *consists of modifications of round switch rules. Initial locations of* TA^{rd} *are the locations from* $\mathcal{B} \subseteq \mathcal{L}$.

For a $\mathsf{TA}^{\mathrm{rd}}$ and a $k \in \mathbb{N}_0$ we define a counter system $\mathsf{Sys}^k(\mathsf{TA}^{\mathrm{rd}})$ as the tuple (Σ^k, I^k, R^k). A configuration is a tuple $\sigma = (\vec{\boldsymbol{\kappa}}, \vec{g}, \mathbf{p}) \in \Sigma^k$, where $\sigma.\vec{\boldsymbol{\kappa}} : \mathcal{D} \to \mathbb{N}_0$ defines values of the counters, for $\mathcal{D} = (\mathcal{L} \times \{k\}) \cup (\mathcal{B}' \times \{k+1\})$; and $\sigma.\vec{g} : \Gamma \times \{k\} \to \mathbb{N}_0$ defines shared variable values; and $\sigma.\mathbf{p} \in \mathbb{N}_0^{|\Pi|}$ is a vector of parameter values.

Note that by using $\mathcal{D}$ in the definition of $\sigma.\vec{\boldsymbol{\kappa}}$ above, every configuration $\sigma \in \mathsf{Sys}^k(\mathsf{TA}^{\mathrm{rd}})$ can be extended to a valid configuration of $\mathsf{Sys}_\infty(\mathsf{TA})$, by assigning zero to all other counters and global variables. In the following, we identify a configuration in $\mathsf{Sys}^k(\mathsf{TA}^{\mathrm{rd}})$ with its extension in $\mathsf{Sys}_\infty(\mathsf{TA})$, since they have the same labeling function λ_k, for every $k \in \mathbb{N}_0$.

We define $\Sigma^k_{\mathcal{B}} \subseteq \Sigma^k$, for a $k \in \mathbb{N}_0$, to be the set of all configurations σ where every process is in a location from $\mathcal{B}$, and all shared variables are set to zero in k, formally, $\sigma.\vec{g}[x, k] = 0$ for all $x \in \Gamma$, and $\sum_{\ell \in \mathcal{B}} \sigma.\vec{\boldsymbol{\kappa}}[\ell, k] = N(\mathbf{p})$, and $\sigma.\vec{\boldsymbol{\kappa}}[\ell, i] = 0$ for all $(\ell, i) \in \mathcal{D} \setminus (\mathcal{B} \times \{k\})$. We call these configurations *border configurations for* k. The set of initial states I^k is a subset of $\Sigma^k_{\mathcal{B}}$.

We define the transition relation R as in $\mathsf{Sys}_\infty(\mathsf{TA})$, *i.e.*, two configurations are in the relation R^k if and only if they (or more precisely, their above described extensions) are in R.

For every $k \in \mathbb{N}_0$ and every $\sigma \in \Sigma_{\mathcal{B}}^k$, there is a corresponding configuration $\sigma' \in \Sigma_{\mathcal{B}}^0$ obtained from σ by renaming the round k to round 0. Let f_k be the renaming function, *i.e.*, $\sigma' = f_k(\sigma)$. Let us define $\Sigma^u \subseteq \Sigma_{\mathcal{B}}^0$ to be the union of all renamed initial configurations from all rounds, that is, $\{f_k(\sigma) \colon k \in \mathbb{N}_0, \sigma \in I^k\}$.

▶ **Theorem 6.** *Let* TA *be non-blocking, and let all fair executions of* $\mathsf{Sys}^0(\mathsf{TA}^{rd})$ *w.r.t.* $\Sigma_{\mathcal{B}}^0$ *terminate. Given a formula* $\varphi[i]$ *from* $\mathsf{LTL}_{\text{-}X}$ *over* AP_i*, for a round variable* i*, we have*
(A) $\mathsf{Sys}^0(\mathsf{TA}^{rd}) \models \mathbf{E}\,\varphi[0]$ *w.r.t. initial configurations* Σ^u *if and only if*
(B) *there exists* $k \in \mathbb{N}_0$ *such that* $\mathsf{Sys}_\infty(\mathsf{TA}) \models \mathbf{E}\,\varphi[k]$.

Proof sketch. The theorem is proved using the following arguments. In statement (B), the existential quantification over k corresponds to the defintion of Σ^u as union, over all rounds, of projections of all reachable initial configurations of that round.

Implication $(B) \to (A)$ exploits the fact that all rounds are equivalent up to renaming of round numbers (with the exception of possible initial configurations).

Implication $(A) \to (B)$, note that an initial configuration in Σ^u is not necessarily initial in round 0, so that one cannot *a priori* take $k = 0$. Let us explain how to extend an execution of round k into an infinite execution in $\mathsf{Sys}_\infty(\mathsf{TA})$. By termination, all rounds up to $k{-}1$ terminate, so that there is execution that reaches a configuration where all processes are in initial locations of round k. The executions or round k mimick the ones of round 0 (modulo the round number). Finally, the non-blocking assumption is required to be always able to extend to infinite executions after round k is terminated. ◀

In Section 4 we showed how to reduce our specifications to formulas of the form $\forall k \in \mathbb{N}_0.\ \mathbf{A}\,\psi[k]$. Theorem 6 deals with negations of such formulas, i.e., with existence of a round k such that formula $\mathbf{E}\,\varphi[k]$ holds. Thus, we can check specifications on the single-round system.

7 Probabilistic Wait-Free Termination

We start by defining two conditions that are sufficient to establish Probabilistic Wait-Free Termination under round-rigid adversaries. Condition (C1) states the existence of a positive probability lower-bound for all processes ending round k with equal final values. Condition (C2) states that if all correct processes start round k with the same value, then they all will decide on that value in that round.

(C1) There is a bound $p \in (0,1]$, such that for every round-rigid adversary a, and every $k \in \mathbb{N}_0$, and every configuration σ_k with parameters $\mathbf{p}$ that is initial for round k, it holds that

$$\mathbb{P}_{\mathsf{a}}^{\sigma_k}\Big(\bigvee\nolimits_{v\in\{0,1\}} \mathbf{G}\,\big(\bigwedge\nolimits_{\ell\in\mathcal{F}_v} \vec{\boldsymbol{\kappa}}[\ell,k]=0\big)\Big) > p.$$

(C2) For every $v \in \{0,1\}$, $\forall k \in \mathbb{N}_0.\ \mathbf{A}\Big(\mathbf{G}\,\bigwedge_{\ell\in\mathcal{I}_{1-v}} \vec{\boldsymbol{\kappa}}[\ell,k]=0 \ \to\ \mathbf{G}\,\bigwedge_{\ell'\in\mathcal{F}\setminus\mathcal{D}_v} \vec{\boldsymbol{\kappa}}[\ell',k]=0\Big)$.
Combining (C1) and (C2), under every round-rigid adversary, from any initial configuration of round k, the probability that all correct processes decide before end of round $k{+}1$ is at least p. Thus the probability not to decide within $2n$ rounds is at most $(1-p)^n$, which tends to 0 when n tends to infinity. This reasoning follows the arguments of [1].

▶ **Proposition 7.** *If* $\mathsf{Sys}_\infty(\mathsf{PTA}) \models (C1) \wedge (C2)$*, then* $\mathsf{Sys}_\infty(\mathsf{PTA}) \models (3)$.

Observe (C2) is a non-probabilistic property of the same form as (5), so that we can check (C2) using the method of Section 6.

In the rest of this section, we detail how to reduce the verification of (C1), to a verification task that can be handled by ByMC. First observe that (C1) contains a single round variable, and recall that we restrict to round-rigid adversaries, so that it is sufficient to check them (omitting the round variables) on the single-round system. We introduce analogous objects as in the non-probabilistic case: $\mathsf{PTA}^{\mathrm{rd}}$ (analogously to Definition 5), and its counter system $\mathsf{Sys}(\mathsf{PTA}^{\mathrm{rd}})$.

7.1 Reducing probabilistic to non-probabilistic specifications

Since probabilistic transitions end in final locations, they cannot appear on a cycle in $\mathsf{PTA}^{\mathrm{rd}}$. Therefore, for fixed parameter valuation $\mathbf{p}$, any path contains at most $N(\mathbf{p})$ probabilistic transitions, and its probability is therefore uniformly lower-bounded. As a consequence:

▶ **Lemma 8.** *Let $\mathbf{p} \in \mathbf{P}_{RC}$ be a parameter valuation. In $\mathsf{Sys}(\mathsf{PTA}^{rd})$, for every LTL formula φ over atomic proposition* AP*, the following two statements are equivalent:*
(a) $\exists p > 0,\ \forall \sigma \in I_{\mathbf{p}},\ \forall \mathsf{a} \in \mathcal{A}^R.\quad \mathbb{P}^{\sigma}_{\mathsf{a}}(\varphi) > p,$
(b) $\forall \sigma \in I_{\mathbf{p}},\ \forall \mathsf{a} \in \mathcal{A}^R,\ \exists \pi \in \mathsf{paths}(\sigma, \mathsf{a}).\quad \pi \models \varphi.$

7.2 Verifying (C1) on a non-probabilistic TA

Applying Lemma 8, proving (C1) is equivalent to proving the following property on $\mathsf{Sys}(\mathsf{PTA}^{\mathrm{rd}})$

$$\forall \sigma \in I_{\mathbf{p}},\ \forall \mathsf{a} \in \mathcal{A}^{\mathrm{R}},\ \exists \pi \in \mathsf{paths}(\sigma, \mathsf{a}).\quad \pi \models \bigvee_{v\in\{0,1\}} \mathbf{G}\,(\bigwedge_{\ell\in\mathcal{F}_v} \vec{\boldsymbol{\kappa}}[\ell] = 0). \tag{6}$$

In the sequel, we explain how to reduce the verification of (6) to checking the simpler formula $\mathbf{A} \bigvee_{v\in\{0,1\}} \mathbf{G}\,(\bigwedge_{\ell\in\mathcal{F}_v} \vec{\boldsymbol{\kappa}}[\ell] = 0)$ on a single-round non-probabilistic TA obtained from $\mathsf{PTA}^{\mathrm{rd}}$.

As in Section 6, it is possible to modify $\mathsf{PTA}^{\mathrm{rd}}$ into a non-probabilistic TA, by replacing probabilistic choices by non-determinism. Still, the quantifier alternation of (6) (universal over initial configurations and adversaries vs. existential on paths) is not in the fragment handled by ByMC [13]. Once an initial configuration σ and an adversary a are fixed, the remaining branching is induced by non-Dirac transitions. By assumption, these transitions lead to final locations only, to both $\mathcal{F}_0$ and $\mathcal{F}_1$, and under round-rigid adversaries, they are the last transitions to be fired. To prove (6), it is sufficient to prove that all processes that fire only Dirac transitions will reach final locations of the same type ($\mathcal{F}_0$ or $\mathcal{F}_1$). If this is the case, then the existence of a path corresponds to all non-Dirac transitions being resolved in the same way. This allows us to remove the non-Dirac transitions from the model as follows. Given a $\mathsf{PTA}^{\mathrm{rd}}$, we define a threshold automaton TA^{m} with locations $\mathcal{L}$ (without $\mathcal{B}'$) such that for every non-Dirac rule $r = (\mathit{from}, \delta_{to}, \varphi, \vec{u})$ in PTA, all locations ℓ with $\delta_{to}(\ell) > 0$ are merged into a new location ℓ^{mrg} in TA^{m}. Note that this location must belong to $\mathcal{F}$. Naturally, instead of a non-Dirac rule r we obtain a Dirac rule $(\mathit{from}, \ell^{\mathrm{mrg}}, \varphi, \vec{u})$. Also we add self-loops at all final locations. Paths in $\mathsf{Sys}(\mathsf{TA}^{\mathrm{m}})$ correspond to prefixes of paths in $\mathsf{Sys}(\mathsf{PTA}^{\mathrm{rd}})$. In $\mathsf{Sys}(\mathsf{TA}^{\mathrm{m}})$, from a configuration σ, an adversary a yields a unique path, that is, $\mathsf{paths}(\sigma, \mathsf{a})$ is a singleton set. Thus, the existential quantifier from (6) can be replaced by the universal one.

By construction, property (6) on $\mathsf{PTA}^{\mathrm{rd}}$ is equivalent to $\mathbf{A} \bigvee_{v\in\{0,1\}} \mathbf{G}\,(\bigwedge_{\ell\in\mathcal{F}_v} \vec{\boldsymbol{\kappa}}[\ell] = 0)$ on $\mathsf{Sys}(\mathsf{TA}^{\mathrm{m}})$. The latter can be checked automatically by ByMC, allowing us to prove (C1).

8 Experiments

We have applied the approach presented in Sections 4–7 to five randomized fault-tolerant distributed algorithms. (The benchmarks and the instructions on running the experiments are available from: `https://forsyte.at/software/bymc/artifact42/`.)

Table 1 Temporal properties verified in our experiments for value 0.

Label	Name	Automaton	Formula
S1	`agreement_0`	N	$\mathbf{A}\,\mathbf{G}\,(\neg\textsc{Ex}\{\mathsf{D0}\}) \vee \mathbf{G}\,(\neg\textsc{Ex}\{\mathsf{D1},\mathsf{E1}\})$
S2	`validity_0`	N	$\mathbf{A}\,\textsc{All}\{\mathsf{V0}\} \to \mathbf{G}\,(\neg\textsc{Ex}\{\mathsf{D1},\mathsf{E1}\})$
S3	`completeness_0`	N	$\mathbf{A}\,\textsc{All}\{\mathsf{V0}\} \to \mathbf{G}\,(\neg\textsc{Ex}\{\mathsf{D1},\mathsf{E1}\})$
S4	`round-term`	N	$\mathbf{A}\,\mathit{fair} \to \mathbf{F}\,\textsc{All}\{\mathsf{D0},\mathsf{D1},\mathsf{E0},\mathsf{E1},\mathsf{CT}\}$
S5	`decide-or-flip`	P	$\mathbf{A}\,\mathit{fair} \to \mathbf{F}\,(\textsc{All}\{\mathsf{D0},\mathsf{E0},\mathsf{CT}\} \vee \textsc{All}\{\mathsf{D1},\mathsf{E1},\mathsf{CT}\})$
S1'	`sim-agreement`	N	$\mathbf{A}\,\mathbf{G}\,(\neg\textsc{Ex}\{\mathsf{D0},\mathsf{E0}\} \vee \neg\textsc{Ex}\{\mathsf{D1},\mathsf{E1}\})$
S1"	`2-agreement`	N	$\mathbf{A}\,\mathbf{G}\,(\neg\textsc{Ex}\{\mathsf{D0},\mathsf{E0}\} \vee \neg\textsc{Ex}\{\mathsf{D1},\mathsf{E1}\} \vee \neg\textsc{Ex}\{\mathsf{D2},\mathsf{E2}\})$

1. Protocol 1 for randomized consensus by Ben-Or [3], with two kinds of crashes: clean crashes (ben-or-cc), for which a process either sends to all processes or none, and dirty crashes (ben-or-dc), for which a process may send to a subset of processes. This algorithm works correctly when $n > 2t$.
2. Protocol 2 for randomized Byzantine consensus (ben-or-byz) by Ben-Or [3]. This algorithm tolerates Byzantine faults when $n > 5t$.
3. Protocol 2 for randomized consensus (rabc-c) by Bracha [5]. It runs as a high-level algorithm together with a low-level broadcast that turns Byzantine faults into "little more than fail-stop (faults)". We check only the high-level algorithm for clean crashes.
4. k-set agreement for crash faults (kset) by Raynal [21], for $k = 2$. This algorithm works in presense of clean crashes when $n > 3t$.
5. Randomized Byzantine one-step consensus (rs-bosco) by Song and van Renesse [23]. This algorithm tolerates Byzantine faults when $n > 3t$, and it terminates fast when $n > 7t$ or $n > 5t$ and $f = 0$.

Following the reduction approach of Sections 4–7, for each benchmark, we have encoded two versions of one-round threshold automata: an N-automaton that models a coin toss by a non-deterministic choice in a coin-toss location, and is used for the non-probabilistic reasoning, and a P-automaton that never leaves the coin-toss location and which is used to prove probabilistic wait-free termination. Both automata are given as the input to Byzantine Model Checker (ByMC) [13], which implements the parameterized model checking techniques for safety [11] and liveness [10] of counter systems of threshold automata.

Both automata follow the pattern shown in Figure 2: They start in one of the initial locations (e.g., V_0 or V_1), progress by switching locations and incrementing shared variables and end up in a location that corresponds to a decision (e.g., D_0 or D_1), an estimate of a decision (e.g., E_0 or E_1), or a coin toss (CT).

Table 1 summarizes the temporal properties that were verified in our experiments. Given the set of all possible locations $\mathcal{L}$, a set $Y = \{\ell_1, \dots, \ell_m\} \subseteq \mathcal{L}$ of locations, and the designated crashed location $\mathsf{CR} \in \mathcal{L}$, we use the shorthand notation: $\textsc{Ex}\{\ell_1, \dots, \ell_\mathsf{m}\}$ for $\bigvee_{\ell \in Y} \vec{\boldsymbol{\kappa}}[\ell] \neq 0$ and $\textsc{All}\{\ell_1, \dots, \ell_\mathsf{m}\}$ for $\bigwedge_{\ell \in \mathcal{L} \setminus Y} (\vec{\boldsymbol{\kappa}}[\ell] = 0 \vee \ell = \mathsf{CR})$. For rs-bosco and kset, instead of checking S1, we check S1' and S1".

Table 2 presents the computational results of our experiments: column $|\mathcal{L}|$ shows the number of automata locations, column $|\mathcal{R}|$ shows the number of automata rules, column $|\mathcal{S}|$ shows the number of SMT queries (which depends on the structure of the automaton and the specification), column *time* shows the computation times – either in seconds or in the format `HH:MM`. As the N-automata have more rules than the P-automata, column $|\mathcal{R}|$ shows the figures for N-automata. To save space, we omit the figures for memory use from the table: Benchmarks 1–5 need 30–170 MB, whereas rs-bosco needs up to 1.5 GB per CPU.

Table 2 The experiments for first 5 rows were run on a single computer (Apple MacBook Pro 2018, 16GB). The experiments for last row (rs-bosco) were run in Grid5000 on 32 nodes (2 CPUs Intel Xeon Gold 6130, 16 cores/CPU, 192GB). Wall times are given.

Automaton			**S1/S1'/S1"**		**S2**		**S3**		**S4**		**S5**	
Name	$\|\mathcal{L}\|$	$\|\mathcal{R}\|$	$\|\mathcal{S}\|$	**Time**	$\|\mathcal{S}\|$	**Time**	$\|\mathcal{S}\|$	**Time**	$\|\mathcal{S}\|$	**Time**	$\|\mathcal{S}\|$	**Time**
ben-or-cc	10	27	9	1	5	0	5	0	5	0	5	0
ben-or-dc	10	32	9	1	5	1	5	0	5	0	5	1
ben-or-byz	9	18	3	1	2	0	2	0	2	0	2	1
rabc-cr	11	31	9	0	5	1	5	1	5	0	5	0
kset	13	58	65	3	65	17	65	12	65	39	65	40
rs-bosco	19	48	156M	3:21	156M	3:02	156M	3:21	TO	TO	156M	3:43

The benchmark rs-bosco is a challenge for the technique of [10]: Its threshold automaton has 12 threshold guards that can change their values almost in any order. Additional combinations are produced by the temporal formulas. Although ByMC reduces the number of combinations by analyzing dependencies between the guards, it still produces between 11! and 14! SMT queries. Hence, we ran the experiments for rs-bosco on 1024 CPU cores of Grid5000 and gave the wall time results in Table 2. (To find the total computing time, multiply wall time by 1024.) ByMC timed out on the property S4 after 1 day (shown as TO).

For all the benchmarks in Table 2, ByMC has reported that the specifications hold. By changing $n > 3t$ to $n > 2t$, we found that rabc-cr can handle more faults (the original $n > 3t$ was needed to implement the underlying communication structure which we assume given in the experiments). In other cases, whenever we changed the parameters, that is, increased the number of faults beyond the known bound, the tool reported a counterexample.

9 Conclusions

Our proof methodology for almost sure termination applies to round-rigid adversaries only. As future work we shall prove that verifying almost-sure termination under round-rigid adversaries is sufficient to prove it for more general adversaries. Transforming an adversary into a round-rigid one while preserving the probabilistic properties over the induced paths, comes up against the fact that, depending on the outcome of a coin toss in some step at round k, different rules may be triggered later for processes in rounds less than k.

A few contributions address automated verification of probabilistic parameterized systems [22, 4, 20, 19]. In contrast to these, our processes are not finite-state, due to the round numbers and parameterized guards. The seminal work by Pnueli and Zuck [22] requires shared variables to be bounded and cannot use arithmetic thresholds different from 1 and n. Algorithms for well-structured transition systems [4] do not directly apply to multi-parameter systems produced by probabilistic threshold automata. Regular model checking [20, 19] cannot handle arithmetic resilience conditions such as $n > 3t$, nor unbounded shared variables.

References

1 Marcos Aguilera and Sam Toueg. The correctness proof of Ben-Or's randomized consensus algorithm. *Distributed Computing*, pages 1–11, 2012. online first.

2 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

3 Michael Ben-Or. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols (Extended Abstract). In *PODC*, pages 27–30, 1983.

4 Nathalie Bertrand and Paulin Fournier. Parameterized Verification of Many Identical Probabilistic Timed Processes. In *FSTTCS*, volume 24 of *LIPIcs*, pages 501–513, 2013.

5 Gabriel Bracha. Asynchronous Byzantine Agreement Protocols. *Inf. Comput.*, 75(2):130–143, 1987.

6 Mouna Chaouch-Saad, Bernadette Charron-Bost, and Stephan Merz. A Reduction Theorem for the Verification of Round-Based Distributed Algorithms. In *RP*, volume 5797 of *LNCS*, pages 93–106, 2009.

7 Andrei Damian, Cezara Drăgoi, Alexandru Militaru, and Josef Widder. Communication-closed asynchronous protocols. In *CAV*, 2019. (to appear).

8 Tzilla Elrad and Nissim Francez. Decomposition of Distributed Programs into Communication-Closed Layers. *Sci. Comput. Program.*, 2(3):155–173, 1982.

9 Michael J. Fischer, Nancy A. Lynch, and M. S. Paterson. Impossibility of Distributed Consensus with one Faulty Process. *J. ACM*, 32(2):374–382, 1985.

10 Igor Konnov, Marijana Lazić, Helmut Veith, and Josef Widder. A Short Counterexample Property for Safety and Liveness Verification of Fault-tolerant Distributed Algorithms. In *POPL*, pages 719–734, 2017.

11 Igor Konnov, Marijana Lazic, Helmut Veith, and Josef Widder. Para^2: Parameterized Path Reduction, Acceleration, and SMT for Reachability in Threshold-Guarded Distributed Algorithms. *Formal Methods in System Design*, 51(2):270–307, 2017.

12 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.

13 Igor Konnov and Josef Widder. ByMC: Byzantine model checker. In *ISoLA (3)*, volume 11246 of *LNCS*, pages 327–342. Springer, 2018.

14 Jure Kukovec, Igor Konnov, and Josef Widder. Reachability in Parameterized Systems: All Flavors of Threshold Automata. In *CONCUR*, pages 19:1–19:17, 2018.

15 Marta Z. Kwiatkowska and Gethin Norman. Verifying Randomized Byzantine Agreement. In *FORTE*, pages 194–209, 2002.

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

17 Marta Z. Kwiatkowska, Gethin Norman, and Roberto Segala. Automated Verification of a Randomized Distributed Consensus Protocol Using Cadence SMV and PRISM. In *CAV*, pages 194–206, 2001.

18 Daniel J. Lehmann and Michael O. Rabin. On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers Problem. In *POPL*, pages 133–138, 1981.

19 Ondrej Lengál, Anthony Widjaja Lin, Rupak Majumdar, and Philipp Rümmer. Fair Termination for Parameterized Probabilistic Concurrent Systems. In *TACAS*, volume 10205 of *LNCS*, pages 499–517, 2017. `doi:10.1007/978-3-662-54577-5_29`.

20 Anthony Widjaja Lin and Philipp Rümmer. Liveness of Randomised Parameterised Systems under Arbitrary Schedulers. In *CAV*, volume 9780 of *LNCS*, pages 112–133. Springer, 2016. `doi:10.1007/978-3-319-41540-6_7`.

21 Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Randomized k-set agreement in crash-prone and Byzantine asynchronous systems. *Theor. Comput. Sci.*, 709:80–97, 2018.

22 Amir Pnueli and Lenore D. Zuck. Verification of Multiprocess Probabilistic Protocols. *Distributed Computing*, 1(1):53–72, 1986. `doi:10.1007/BF01843570`.

23 Yee Jiun Song and Robbert van Renesse. Bosco: One-Step Byzantine Asynchronous Consensus. In *DISC*, volume 5218 of *LNCS*, pages 438–450, 2008.

A Sound Foundation for the Topological Approach to Task Solvability

Jérémy Ledent
École Polytechnique, Palaiseau, France
jeremy.ledent@lix.polytechnique.fr

Samuel Mimram
École Polytechnique, Palaiseau, France
samuel.mimram@lix.polytechnique.fr

Abstract

The area of fault-tolerant distributed computability is concerned with the solvability of decision tasks in various computational models where the processes might crash. A very successful approach to prove impossibility results in this context is that of combinatorial topology, started by Herlihy and Shavit's paper in 1999. They proved that, for wait-free protocols where the processes communicate through read/write registers, a task is solvable if and only if there exists some map between simplicial complexes satisfying some properties. This approach was then extended to many different contexts, where the processes have access to various synchronization and communication primitives. Usually, in those cases, the existence of a simplicial map from the protocol complex to the output complex is taken as the definition of what it means to solve a task. In particular, no proof is provided of the fact that this abstract topological definition agrees with a more concrete operational definition of task solvability. In this paper, we bridge this gap by proving a version of Herlihy and Shavit's theorem that applies to any kind of object. First, we start with a very general way of specifying concurrent objects, and we define what it means to implement an object B in a computational model where the processes are allowed to communicate through shared objects $A_1, \ldots, A_k$. Then, we derive the notion of a decision task as a special case of concurrent object. Finally, we prove an analogue of Herlihy and Shavit's theorem in this context. In particular, our version of the theorem subsumes all the uses of the combinatorial topology approach that we are aware of.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases Fault-tolerant protocols, Asynchronous computability, Combinatorial topology, Protocol complex, Distributed task

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.34

1 Introduction

The typical framework in which one studies distributed computing is that of asynchronous processes communicating through shared objects. A wide variety of computational models have been introduced, depending on the communication primitives the processes are allowed to use, and the assumptions made on the kind of failures that might happen during the computation. The area of fault-tolerant computability [6] studies what kind of *decision tasks* can be solved in such models. To solve a task, each process starts with a private input value, and after communicating with the other processes, it has to decide on an output. For instance, a well-known task is *consensus*, where the processes must agree on one of their inputs.

A very well-studied setting is the one of wait-free protocols communicating through shared read/write registers. In order to prove impossibility results in this context, people started developing powerful mathematical tools based on algebraic topology [1, 13]. The fundamental paper by Herlihy and Shavit [10] provides a topological characterization of the tasks that can be solved by a wait-free protocol using read/write registers: they proved the *asynchronous*

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 34; pp. 34:1–34:15
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

computability theorem, which says that a read/write protocol solves a decision task if and only if there exists a map from a subdivision of the input complex to the output complex, which is *carried* by the task specification. The subdivided simplicial complex that appears in this theorem was the first occurrence of a *protocol complex*, a very compact way of expressing combinatorially the partial knowledge that each process acquires during the computation.

Soon thereafter, it became clear that this method actually extends beyond the setting of read/write registers: many other computational models can also be described as protocol complexes. This idea gave birth to the *combinatorial topology* approach to distributed computing [6]. Generalizing the ideas behind the asynchronous computability theorem, one can define an abstract notion of protocol using so-called *carrier maps* between an input complex and a protocol complex. Then, we can take Herlihy and Shavit's characterization as the *definition* of what it means for a protocol to solve a task. This abstract approach is very appealing because it offers a great deal of generality: for example, the set-agreement task cannot be solved in any computational model whose protocol complex is a pseudomanifold [6, Chapter 9]. We are thus able to prove impossibility results for a wide class of computational models, instead of studying them one at a time. In principle, most computational models and synchronization primitives can be formulated using the protocol complex formalism: it has been used in the literature to model processes communicating using test-and-set objects [8], (N, k)-consensus objects [7], weak symmetry breaking and renaming [4], or various synchronous or asynchronous message passing primitives [9].

The drawback of this high level of abstraction is that we have to *trust* that our protocol complex faithfully represents the behavior of the objects that we want to model. When we want to define, for example, the protocol complex which is supposed to model test-and-set computation, we must define it carefully so that the simplicial notion of "solving" a task will agree with the concrete operational semantics of test-and-set objects. Ideally, one would have to prove another version of the asynchronous computability theorem for test-and-set protocols, relating a concrete notion of solvability with the abstract simplicial definition. This is usually not done in practice, since it is often clear intuitively that the proposed notion of protocol complex makes sense operationally.

In this paper, we generalize the asynchronous computability theorem to a large class of concurrent objects. In Section 2, we define the "concrete" notion of solvability that we will later prove equivalent to the simplicial one. Our goal is to model processes which communicate though arbitrary shared objects. In particular, our notion of object is general enough to include all the objects mentioned above. Our notion of object specification is discussed more thoroughly in [5] where, in particular, we prove that it is equivalent to specifications based on interval-linearizability [2]. In Section 3, we discuss the notion of task and compare it to the notion of one-shot object. It had been remarked recently that tasks are less expressive than one-shot objects [2]. We explore more in depth the relationship between the two, and as a result we obtain a characterization of the one-shot objects that correspond to tasks. Finally, in Section 4, we state and prove our generalized asynchronous computability theorem. It encompasses the original theorem of Herlihy and Shavit for read/write registers, as well as all the other variants that have been implicitly used in the literature.

2 A computational model based on trace semantics

In this section, we define a concrete semantics for protocols where the processes communicate through shared objects. What we mean here by "concrete" is that it is based on interleavings of execution traces, as opposed to the abstract topological semantics of Section 4. The formalism that we use here (i.e., all of Section 2) was developed and discussed more thoroughly in a previous article [5]. It is quite similar to other trace-based operational semantics found in the literature [12, 3]. There are mainly two things that we need to define: in Section 2.1, we give a very general way of specifying concurrent objects, and in Section 2.2, we define what it means to implement a new object B, assuming we have access to objects $A_1, \dots, A_k$.

2.1 Specifying concurrent objects

Our goal here is to define a notion of concurrent object which includes all the objects that are relevant to fault-tolerant computability. As it was remarked in [2], the tasks studied in this field cannot be specified using common techniques such as linerizability [11]. Thus, they introduced the notion of *interval-linearizability* in order to unify tasks and objects. The definition that we use here was introduced in [5] and shown to be equivalent to interval-linearizability. In the rest of the paper, we suppose fixed a number $n \in \mathbb{N}$ of *processes* and write $[n] = \{0, 1, \dots, n-1\}$ for the set of *process names* (a process is identified by its number). We also suppose fixed a set $\mathcal{V}$ of *values* which can be exchanged between processes and objects (typically $\mathcal{V} = \mathbb{N}$). The set $\mathcal{A}$ of possible *actions* for an object is defined as

$$\mathcal{A} \quad = \quad \{\mathsf{i}_i^x \mid i \in [n], x \in \mathcal{V}\} \cup \{\mathsf{r}_i^y \mid i \in [n], y \in \mathcal{V}\}$$

An action is thus either

- i_i^x: an *invocation* of the object by the process i with input value x,
- r_i^y: a *response* of the object to the process i with output value y.

An *execution trace* is a finite sequence of actions, i.e., an element of $\mathcal{A}^*$; we write ε for the empty trace and $T \cdot T'$ for the concatenation of two traces T and T'. Given a process $i \in [n]$, the i-th *projection* $\pi_i(T)$ of a trace $T \in \mathcal{T}$ is the trace obtained from T by removing all the actions of the form i_j^x or r_j^x with $j \neq i$. A trace $T \in \mathcal{T}$ is *alternating* if for all $i \in [n]$, $\pi_i(T)$ is either empty or it begins with an invocation and alternates between invocations and responses, i.e., using the traditional notation of regular expressions:

$$\pi_i(T) \quad \in \quad \Big(\bigcup_{x,y \in \mathcal{V}} \mathsf{i}_i^x \cdot \mathsf{r}_i^y\Big)^* \cdot \Big(\bigcup_{x \in \mathcal{V}} \mathsf{i}_i^x + \varepsilon\Big)$$

We write $\mathcal{T} \subseteq \mathcal{A}^*$ for the set of alternating traces. In the remaining of the paper, we will only consider alternating traces, and often drop the adjective "alternating". If $\pi_i(T)$ ends with an invocation, we call it a *pending* invocation. An alternating trace T is *complete* if it does not have any pending invocation.

▶ **Definition 1.** *A* concurrent specification *σ is a subset of $\mathcal{T}$ which is*
(1) prefix-closed*: if $T \cdot T' \in \sigma$ then $T \in \sigma$,*
(2) non-empty*: $\varepsilon \in \sigma$,*
(3) receptive*: if $T \in \sigma$ and $\pi_i(T)$ has no pending invocation, then $T \cdot \mathsf{i}_i^x \in \sigma$ for every $x \in \mathcal{V}$,*
(4) total*: if $T \in \sigma$ and $\pi_i(T)$ has a pending invocation, there exists $x \in \mathcal{V}$ such that $T \cdot \mathsf{r}_i^x \in \sigma$,*
(5) *is closed under* expansion*:*

- *if $T \cdot a_j \cdot \mathsf{i}_i^x \cdot T' \in \sigma$ where a_j is an action of process $j \neq i$, then $T \cdot \mathsf{i}_i^x \cdot a_j \cdot T' \in \sigma$.*
- *if $T \cdot \mathsf{r}_i^y \cdot a_j \cdot T' \in \sigma$ where a_j is an action of process $j \neq i$, then $T \cdot a_j \cdot \mathsf{r}_i^y \cdot T' \in \sigma$.*

We write CSpec *for the set of concurrent specifications.*

A concurrent specification σ is the set of all executions that we consider acceptable: a protocol *implements* the specification σ if all the execution traces that it generates belong to σ (this will be detailed in Section 2.2). The axioms (1-3) are quite natural and commonly considered in the literature (e.g. in [11]). They can be read as follows: (1) an object can do one action at a time, (2) an object can do nothing, (3) it is always possible to invoke an object. The axiom (4) states that objects always answer and in a non-blocking way; this is a less fundamental axiom and more of a design choice, since we want to model wait-free computation. Note that receptivity (3) does not force objects to accept all inputs: we could have a distinguished "error" value which is returned in case of an invalid input. Similarly, an object with several inputs, or several interacting methods (for example, a stack) can be modeled by choosing a suitable set of values $\mathcal{V}$.

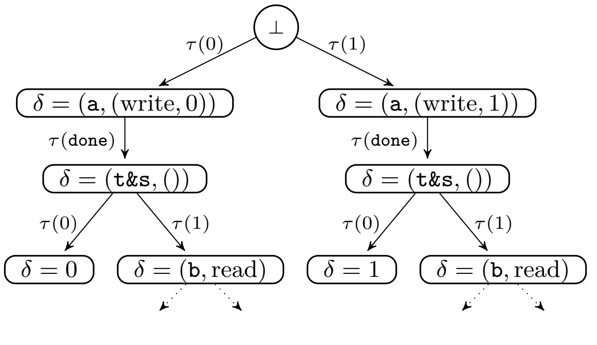
The condition (5) might seem more surprising, and will be crucial to establish the correspondence between tasks and objects in Section 3. Some consequences of this condition, and the reasons why it is a desirable property of concurrent specifications, are discussed more thoroughly in [5]. Intuitively, this condition says that if a given execution is correct, then a similar execution where some process is idle for a while just after invoking, or just before responding, must also be correct, since both executions are indistinguishable. In particular, condition (5) implies that invocations "commute": we have $T \cdot \mathsf{i}_i^x \cdot \mathsf{i}_j^y \cdot T' \in \sigma$ *iff* $T \cdot \mathsf{i}_j^y \cdot \mathsf{i}_i^x \cdot T' \in \sigma$, and similarly for responses. We say that two traces $T, T' \in \mathcal{T}$ are *equivalent*, written $T \equiv T'$, if one is obtained from the other by reordering the actions within each block of consecutive invocations or consecutive responses. Generally, in the rest of the paper, we are only interested in studying traces up to equivalence.

2.2 Program semantics

In this section, we provide an operational model for concurrent programs communicating through shared objects. We assume given a set Obj of objects: they might be, for instance, concurrent data structures that have already been implemented, and that our programs are able to use in order to compute and communicate. We do not want to depend on a particular implementation of these objects, but on their specification. Thus, each object comes with its concurrent specification (as in Definition 1), which is the set of behaviors that it might exhibit. Note that our model does not have any special construct for reading and writing in the shared memory: we assume that the memory itself is given as an object in Obj, with an appropriate specification. Thus, the only meaningful action a program can take is to call one of the objects; and possibly do some local computation to determine what the next call should be.

To abstract away the syntax of the programming language, we use an automata-like representation, which roughly corresponds to the control-flow graph of the program.

```
consensus(v) {
  a.write(v);
  x = t&s();
  if (x == 0)
    return v;
  else
    v' = b.read();
    return v';
}
```



The example above shows an implementation of binary consensus among two processes, using three objects: two read-write registers a and b, and a test-and-set object t&s. In a given state of the automaton, the decision function δ indicates which is the next object that the program will call, and the transition function τ says what the next state will be depending on the return value of the call. The pseudo-code and automaton above represent the program run by one of the two processes, the other process should switch the role of `a` and `b`.

We suppose fixed a set Obj of objects, along with their concurrent specification $\mathsf{spec}(o) \in \mathsf{CSpec}$ for each $o \in \mathsf{Obj}$. Here, a program is basically a piece of code (executed by one of the processes) which takes a value as input, makes several calls to the objects in Obj (using algebraic operations to combine their results) and finally returns a value. Formally,

▶ **Definition 2.** *A* program *is a quadruple* $(Q, \bot, \delta, \tau)$ *consisting of:*

- *a (possibly infinite) set* Q *of* local states *containing an* idle state $\bot \in Q$*,*
- *a* decision function $\delta : Q \setminus \{\bot\} \to (\mathsf{Obj} \times \mathcal{V}) \sqcup \mathcal{V}$*,*
- *a* transition function $\tau : Q \times \mathcal{V} \to Q \setminus \{\bot\}$*.*

The idle state is the one where the program is waiting to be called with some input value x, in which case it will go to state $\tau(\bot, x)$. After that, the decision function gives the next step of the process depending on the current state: either call some object with some input value, or terminate and output some value. In the case where an object is called, the transition function gives the new local state of the process, depending on the previous state and the value returned by the object.

A *protocol* P is given by a program $P_i = (Q_i, \bot_i, \delta_i, \tau_i)$ for each process $i \in [n]$. The *global state* of a protocol P is an element $q = (q_0, \ldots, q_{n-1})$ of $\overline{Q} = \prod_i Q_i$, consisting of a state for each process P_i. The initial state is $q_{\mathrm{init}} = (\bot_0, \ldots, \bot_{n-1})$ and, given a global state q, we write $q[i \leftarrow q_i']$ for the state where the i-th component q_i has been replaced by q_i'. We now describe the set $\mathcal{A}$ of possible actions for P, as well as their effect $\Delta : \overline{Q} \times \mathcal{A} \to \overline{Q}$ on global states.

- i_i^x: the i-th process is called with input value $x \in \mathcal{V}$. The local state q_i of process i is changed from $\bot_i$ to $\tau_i(\bot_i, x)$:

  $$\Delta(q, \mathsf{i}_i^x) \quad = \quad q[i \leftarrow \tau_i(\bot_i, x)]$$

  where the state on the right is q where q_i has been replaced by $\tau_i(\bot_i, x)$.
- $\mathsf{i}(o)_i^x$: the i-th process invokes the object $o \in \mathsf{Obj}$ with input value $x \in \mathcal{V}$. This does not have any effect on the global state:

  $$\Delta(q, \mathsf{i}(o)_i^x) \quad = \quad q$$
- $\mathsf{r}(o)_i^x$: the object $o \in \mathsf{Obj}$ returns some output value $x \in \mathcal{V}$ to the i-th process. The local state of process i is updated according to its transition function τ_i:

  $$\Delta(q, \mathsf{r}(o)_i^x) \quad = \quad q[i \leftarrow \tau_i(q_i, x)]$$
- r_i^x: the i-th process has finished computing, returning the output value $x \in \mathcal{V}$. It goes back to idle state:

  $$\Delta(q, \mathsf{r}_i^x) \quad = \quad q[i \leftarrow \bot_i]$$

The actions of the form i_i^x and r_i^x (resp. $\mathsf{i}(o)_i^x$ and $\mathsf{r}(o)_i^x$) are called *outer* (resp. *inner*) *actions*. Given a trace $T \in \mathcal{A}^*$ and an object o, we denote by T_o, called the *inner projection on* o, the trace obtained from T by keeping only the inner actions of the form $\mathsf{i}(o)_i^x$ or $\mathsf{r}(o)_i^y$. The function Δ is extended as expected as a function $\Delta : \overline{Q} \times \mathcal{A}^* \to \overline{Q}$, i.e., $\Delta(q, T \cdot T') = \Delta(\Delta(q, T), T')$ and $\Delta(q, \varepsilon) = q$. A trace is valid if at each step in the execution, the next action is taken according to the decision function δ. Formally:

▶ **Definition 3.** *A trace $T \in \mathcal{A}^*$ is* valid *when for every strict prefix U of T, writing $T = U \cdot a \cdot V$ and $q = \Delta(q_{\text{init}}, U)$, with $a \in \mathcal{A}$, we have:*

- *if $a = \mathsf{i}_i^x$ then $q_i = \bot_i$,*
- *if $a = \mathsf{r}_i^y$ then $q_i \neq \bot_i$ and $\delta_i(q_i) = y$,*
- *if $a = \mathsf{i}(o)_i^x$ then $q_i \neq \bot_i$ and $\delta_i(q_i) = (o, x)$,*
- *if $a = \mathsf{r}(o)_i^y$ then $q_i \neq \bot_i$.*

Moreover, we require that for every object $o \in \mathsf{Obj}$, the inner projection T_o belongs to $\mathsf{spec}(o)$. The set of valid traces for P is written $\mathcal{T}_P \subseteq \mathcal{A}^*$.

A protocol P is *wait-free* if there is no valid infinite trace (i.e., all its prefixes are valid) involving only inner-i-actions after some position: in other words, a process running alone will eventually decide an output value. Given a trace $T \in \mathcal{A}^*$, we write $\pi(T)$ for the trace obtained by keeping only outer actions.

▶ **Definition 4.** *The* semantics *of a protocol P is the set of traces $[\![P]\!] = \{\pi(T) \mid T \in \mathcal{T}_P\}$ and P* implements *a concurrent specification σ whenever $[\![P]\!] \subseteq \sigma$, i.e., all the outer traces that P can produce are correct with respect to σ.*

An important property that was proved in [5] is that $[\![P]\!]$ itself is a concurrent specification in the sense of Definition 1. Indeed, given any protocol P, we should be able to specify abstractly what P is doing: that specification is precisely $[\![P]\!]$. In particular, since $[\![P]\!]$ satisfies all the axioms of concurrent specifications, we are sure that these axioms are reasonable, in the sense that they are validated in our model.

3 Tasks as a particular kind of one-shot objects

The topological approach to distributed computing [6] is not interested in implementing long-lived objects as in the previous section, but in solving *decision tasks.* In a decision task, each process starts with a private input value, it communicates with the other processes by using some shared objects, and then it must eventually decide an output value. The most well-known example is the *consensus* task, where the processes have to agree on a common output value. Thus, in a decision task, all the processes start the computation together, and once a process has decided an output value, it does not take part in the computation anymore. This is contrasted with concurrent objects, where new processes can start and terminate at any point during the computation, and a single process is allowed to make several consecutive calls to the object.

In this section, we compare tasks and one-shot objects, that is, objects which can be called only once by each process. It was already observed in [2] that tasks are weaker than one-shot objects: some objects cannot be expressed as a task. They defined a notion of “extended task” which is as expressive as one-shot objects. Our goal here is dual: we want to characterize the subclass of one-shot objects which correspond to tasks.

We start by giving a formal definition of tasks. The formalism that we use here is inspired of the one used in Herlihy and Shavit’s paper [10]. Readers familiar with the topological definition of a task should recognize that this a direct reformulation of it. Recall that the set of values is written $\mathcal{V}$ and n is the number of processes. Let $\bot$ be a fresh symbol which is not in $\mathcal{V}$. A *vector* is a tuple $U \in (\mathcal{V} \cup \{\bot\})^n$, such that at least one component of U is not $\bot$. We write U_i for the i-th component of U, and $U[i \leftarrow x]$ for the vector U where the i-th component U_i has been replaced by $x \in \mathcal{V}$. Given two vectors U and V, we say that U *matches* V when $U_i = \bot$ iff $V_i = \bot$. We say that U is a *face* of V, written $U \preceq V$, when for all i, either $U_i = V_i$ or $U_i = \bot$. A vector U is *maximal* if none of its components is $\bot$. If X is a set of vectors, its *downward closure* is $\downarrow X = \{U \mid U \preceq V \text{ for some } V \in X\}$.

▶ **Definition 5.** *A* task *is a triple $\Theta = (I, O, \Delta)$ such that:*

- *$I = \mathcal{V}^n$ and $O \subseteq \mathcal{V}^n$ are sets of maximal vectors. The elements of I (resp. of O) and their faces are called* input vectors *(resp.* output vectors*).*
- *$\Delta \subseteq {\downarrow} I \times {\downarrow} O$ is a relation between input and output vectors, such that:*
  - *Δ only relates matching vectors,*
  - *for every input vector $U \in {\downarrow} I$, $\Delta(U) \neq \varnothing$,*
  - *for all input vectors $U \preceq U'$, $\Delta(U) \subseteq {\downarrow}\Delta(U')$.*

Above, we write $\Delta(U) = \{V \mid (U, V) \in \Delta\}$. Intuitively, the task specification says that if each process i starts with the input U_i, the set $\Delta(U)$ consists of all the output vectors that are considered acceptable outputs. A '$\perp$' component in a vector represents a process which is not participating in the computation, either because it crashed, or because it was so slow that all the other processes terminated before it could take any steps. With that in mind, the third condition on Δ asserts that, if such a "slow" process wakes up, there is at least one valid output which is compatible with what the other processes already decided. Note that, to match the receptivity property of concurrent specifications, we are requiring $I = \mathcal{V}^n$, i.e., a task must specify what the outputs should be given any input vector. This is not a restriction compared to the usual definition of a task: if we do not care about what happens on some of the input vectors, we can simply consider that every output is correct.

We now describe how we can turn a task into a one-shot object. Given a vector U, the set $\{i \in [n] \mid U_i \neq \perp\}$ is called the *participating set* of U. A pair $(U, V) \in \Delta$, where U and V have participating set $I = \{i_1, \ldots, i_k\}$, corresponds to a correct execution trace of the form $\mathsf{i}_{i_1}^{U_{i_1}} \cdots \mathsf{i}_{i_k}^{U_{i_k}} \cdot \mathsf{r}_{i_1}^{V_{i_1}} \cdots \mathsf{r}_{i_k}^{V_{i_k}}$, that is, a trace where all the participating processes invoke with the input values from U, and then they all respond with the output values from V. Remember that the order of consecutive invocations or consecutive responses does not matter. For convenience, we will write such a trace $\mathsf{i}^U \cdot \mathsf{r}^V$.

Thus, the task specification Δ gives us a set of execution traces $\{\mathsf{i}^U \cdot \mathsf{r}^V \mid (U, V) \in \Delta\} \subseteq \mathcal{T}$. But this set does not satisfy the conditions of Definition 1: we need to specify which traces are correct or not, among the traces of other "shapes" (i.e., traces where invocations and responses are interleaved). For example, consider a trace of the form $\mathsf{i}^U \cdot \mathsf{r}^V \cdot \mathsf{i}_j^x \cdot \mathsf{r}_j^y$, where $(U, V) \in \Delta$ have a participating set I, and $j \notin I$. Intuitively, this represents the situation where all the processes in I ran together without hearing from j, and after all of these processes terminated, the process j ran alone with input x and decided output y. What should be the condition on x and y for this trace to be considered correct? The most sensible answer is that we should require $(U[j \leftarrow x], V[j \leftarrow y]) \in \Delta$. The next definition generalizes this idea to traces of any shape.

▶ **Definition 6.** *Let $\Theta = (I, O, \Delta)$ be a task. We define the one-shot concurrent specification $G(\Theta) \subseteq \mathcal{T}$ as the set of execution traces $T \in \mathcal{T}$ such that:*

- *T is one-shot, i.e., every process has at most one invocation and one response in T,*
- *there exists a completion T' of T (obtained by appending responses to the pending invocations), such that for every non-empty prefix S of T' which is complete, we have $(U_S, V_S) \in \Delta$, where the i-th component of U_S (resp. V_S) is x if i_i^x (resp., r_i^x) appears in S, and $\perp$ otherwise.*

▶ **Proposition 7.** *$G(\Theta)$ is a one-shot concurrent specification, i.e., it satisfies the properties of Definition 1, except that we only require receptivity among one-shot traces.*

Proof. Prefix-closure (1) and non-emptiness (2) are obvious.

- (3'): Let $T \in G(\Theta)$ and assume that no action from process i occurs in T. Let $x \in \mathcal{V}$, we want to show that $T \cdot \mathsf{i}_i^x \in G(\Theta)$. Let $T' = T \cdot \widehat{T}$ be the completion of T that appears in the definition of $G(\Theta)$ ($\widehat{T}$ consists of responses to the pending invocations of T). In particular, since T' is complete and a prefix of itself, we have $(U_{T'}, V_{T'}) \in \Delta$.
  What we have to do is find a response to the invocation i_i^x that obeys the specification of the task Θ. Since i does not participate in T, the vector $U_{T'}$ has a $\bot$ at position i, and so $U_{T'} \preceq U_{T'}[i \leftarrow x]$. Thus, we have $\Delta(U_{T'}) \subseteq {\downarrow}\Delta(U_{T'}[i \leftarrow x])$, since Θ is a task. So in particular $V_{T'} \in {\downarrow}\Delta(U_{T'}[i \leftarrow x])$, which means that there is some $W \in \Delta(U_{T'}[i \leftarrow x])$ which extends $V_{T'}$. Let $y \in \mathcal{V}$ be the i-th component of W. Pick the complete trace $T'' = T \cdot \mathsf{i}_i^x \cdot \widehat{T} \cdot \mathsf{r}_i^y$. Then we can check that $(U_{T''}, V_{T''}) = (U_{T'}[i \leftarrow x], W) \in \Delta$, and since all the other complete prefixes of T'' are also complete prefixes of T, we finally get $T \cdot \mathsf{i}_i^x \in G(\Theta)$.
- (4): Let $T \in G(\Theta)$ and assume that $\pi_i(T)$ has a pending invocation. Using the same notations as before, the suffix $\widehat{T}$ must contain a response r_i^y to this invocation. Then just by switching the order of the responses in $\widehat{T}$, we obtain $T \cdot \mathsf{r}_i^y \in G(\Theta)$.
- (5): Let $T = T_1 \cdot a_j \cdot \mathsf{i}_i^x \cdot T_2 \in G(\Theta)$, with $j \neq i$, and take the same notations as in the previous cases. We claim that $T_1 \cdot \mathsf{i}_i^x \cdot a_j \cdot T_2 \cdot \widehat{T}$ satisfies the conditions of Definition 6. The only way that this could fail is if the complete prefix S is $T_1 \cdot \mathsf{i}_i^x$: all other cases are covered by the fact that $T \in G(\Theta)$. But $T_1 \cdot \mathsf{i}_i^x$ cannot be complete since it has a pending invocation. So $T_1 \cdot \mathsf{i}_i^x \cdot a_j \cdot T_2 \in G(\Theta)$.
  The second half of condition (5) is proved similarly. ◀

There is also a map in the other direction: from a one-shot concurrent specification σ, we can produce a task $F(\sigma)$. This direction is much easier to define, all we have to do is keep all the traces of σ which consist of a sequence of invocations followed by a sequence of responses, and put in Δ the corresponding pair (U, V).

▶ **Definition 8.** *Given a one-shot concurrent specification $\sigma \subseteq \mathcal{T}$, the task $F(\sigma) = (I, O, \Delta)$ is defined as follows. For each trace $T \in \sigma$ of the form $T = \mathsf{i}_{i_1}^{x_1} \cdots \mathsf{i}_{i_k}^{x_k} \cdot \mathsf{r}_{i_1}^{y_1} \cdots \mathsf{r}_{i_k}^{y_k}$ (we say that T is* fully-concurrent*), we define the vectors U_T (resp., V_T) whose i_j-th component is x_j (resp., y_j) and all the other components are $\bot$. Then $\Delta = \{(U_T, V_T) \mid T \in \sigma$ is fully-concurrent$\}$, and I (resp., O) is the set of all the maximal input (resp., output) vectors that appear in Δ.*

▶ **Proposition 9.** *$F(\sigma)$ is a task.*

Proof. First, we justify that $\Delta \subseteq {\downarrow}I \times {\downarrow}O$. Let $(U_T, V_T) \in \Delta$ for some trace T. We want to show that U_T and V_T are faces of maximal vectors that appear in Δ. We will find a fully-concurrent trace $T' \in \sigma$ such that $U_T \preceq U_{T'}$ and $V_T \preceq V_{T'}$. For each process i that does not participate in T, by the receptivity property of σ, we can add an invocation i_i^x at the end of T. By totality, this invocation has an appropriate response r_i^y, that we add at the end of the trace. Then by applying the expansion property several times, we can push all the new invocations to the left to obtain the trace $T' \in \sigma$ which is fully-concurrent.

We then check that the three conditions on Δ are satisfied. Δ always relates matching vectors because the trace T has matching invocation and responses. For any input vector U, we can use totality of σ to find matching responses, which shows that $\Delta(U) \neq \varnothing$. Finally, given two input vectors $U \preceq U'$, and an element V of $\Delta(U)$, let T be the trace witnessing that $(U, V) \in \Delta$. Then, as in the first paragraph, we can add the missing invocations by receptivity, and matching responses by totality, and get a fully-concurrent trace by expansion. This yields a pair $(U', V') \in \Delta$, with $V \preceq V'$, which is what we want. ◀

The maps F and G are not inverse of each-other: as we stated in the introduction of the section, one-shot objects are more expressive than tasks (an example of a simple object which cannot be expressed as a task is exhibited in [2]). But the next Theorem shows that we still have a close correspondence between tasks and objects. Given two tasks $\Theta = (I, O, \Delta)$ and $\Theta' = (I', O', \Delta')$, we write $\Theta \subseteq \Theta'$ when $\Delta \subseteq \Delta'$.

▶ **Theorem 10.** *The maps F and G are monotonic, and they form a Galois connection between tasks and one-shot objects: for every task Θ and one-shot specification σ, we have*

$$\sigma \subseteq G(\Theta) \iff F(\sigma) \subseteq \Theta$$

Moreover, the following equality holds: $F \circ G(\Theta) = \Theta$.

Proof. The monotonicity of F and G follows directly from the definitions. In the rest of the proof, we write $F(\sigma) = (I', O', \Delta')$.

- ($\Rightarrow$): Assume $\sigma \subseteq G(\Theta)$. Let $(U_T, V_T) \in \Delta'$, for some fully-concurrent trace $T \in \sigma$. So, we also have $T \in G(\Theta)$. Since T is already complete (and a prefix of itself), we obtain $(U_T, V_T) \in \Delta$.
- ($\Leftarrow$): Conversely, assume $F(\sigma) \subseteq \Theta$, and let $T \in \sigma$. First, we complete T by adding response events to the pending invocations using totality. We get a trace $T' = T \cdot \widehat{T} \in \sigma$. Let S be a prefix of T' which is complete. By prefix-closure, we get $S \in \sigma$. Then, using the expansion property, we can push all the invocations to the left in order to get a trace $S' \in \sigma$ which is fully-concurrent. Moreover, since S' is obtained by reordering the actions of S, we have $U_S = U_{S'}$ and $V_S = V_{S'}$. By definition of $F(\sigma)$, we have $(U_{S'}, V_{S'}) \in F(\sigma)$, so by assumption $(U_S, V_S) = (U_{S'}, V_{S'}) \in G(\Theta)$.
- From the first implication, since $G(\Theta) \subseteq G(\Theta)$, we get $F \circ G(\Theta) \subseteq \Theta$. To prove the other inclusion, take $(U, V) \in \Delta$, and consider the associated trace $T = \mathsf{i}^U \cdot \mathsf{r}^V$. We have (by construction) $U = U_T$ and $V = V_T$, so we just need to check that $T \in G(\Theta)$. But since T is already complete, and the only complete non-empty prefix of T is T itself, with $(U_T, V_T) \in \Delta$, this is the case. ◀

Theorem 10 tells us a lot about the relationship between tasks and one-shot objects. The implication from left to right explains in what sense G is a canonical way to turn a task into a one-shot object: $G(\Theta)$ is the largest one-shot specification whose set of fully-concurrent traces obeys the task Θ. Moreover, the equality $F \circ G = \mathsf{id}$, tells us that G is an injection of tasks into one-shot objects. Thus, the objects that we are interested in are precisely the ones in the image of G.

▶ **Definition 11.** *A* task object *σ is a one-shot concurrent specification which can be written as $\sigma = G(\Theta)$ for some task Θ.*

The maps F and G form a bijection between tasks and task objects: indeed, given a task object $\sigma = G(\Theta)$, we have $G \circ F(\sigma) = G \circ F \circ G(\Theta) = G(\Theta) = \sigma$. Conversely, if a one-shot object σ satisfies $\sigma = G \circ F(\sigma)$, then it is a task object (corresponding to the task $F(\sigma)$). Thus, we have a characterization of task objects, which does not refer to the notion of task: σ is a task object *iff* $G \circ F(\sigma) = \sigma$. In fact, since the inclusion $\sigma \subseteq G \circ F(\sigma)$ holds for any one-shot object (as a consequence of Theorem 10), we even have the equivalence: σ is a task object *iff* $G \circ F(\sigma) \subseteq \sigma$. If we reformulate this inclusion by unfolding the definitions of F and G, we obtain a kind of closure property, in the style of Definition 1:

(6) *Task property:* for every complete one-shot trace $T \in \mathcal{T}$, if for every complete prefix S of T, expanding S gives a fully-concurrent trace $S' \in \sigma$, then $T \in \sigma$.

Then, a task object is a set of one-shot traces that satisfies the axioms (1) – (6).

4 An asynchronous computability theorem for arbitrary objects

The *asynchronous computability theorem* of Herlihy and Shavit [10] states that a task Θ has a wait-free protocol using read/write registers if and only if there exists a chromatic subdivision of the input complex, and a chromatic simplicial map from this subdivision to the output complex, that satisfies some conditions. That statement actually combines two claims: (a) a given protocol P using read/write registers solves the task Θ if and only if there exists a chromatic simplicial map from the protocol complex of P to the output complex, satisfying some conditions; and (b) to study task solvability using read/write registers, we can restrict to a particular class of protocols (iterated immediate snapshots) whose protocol complexes are subdivisions of the input complex.

In this section, we will prove a generalization of claim (a), which works not only for read/write registers, but for protocols which are allowed to use any combination of arbitrary objects, as defined in Section 2.2. We will not have a counterpart of claim (b), since this characterization is specific to the particular case of protocols using read/write registers.

4.1 Preliminaries

We first recall the definitions from combinatorial topology that we will be using. A more thorough account of these notions can be found in [6]. A *simplicial complex* $\mathcal{C} = (V, S)$ consists of a set V of *vertices* and a non-empty set S of finite subsets of V called *simplices*, such that:

- for each vertex $v \in V$, $\{v\} \in S$, and
- S is closed under containment, i.e., if $X \in S$ and $Y \subseteq X$, then $Y \in S$.

We sometimes abuse notations and write $X \in \mathcal{C}$ instead of $X \in S$ to mean that X is a simplex of $\mathcal{C}$. If $Y \subseteq X$, we say that Y is a *face* of X. The simplices which are maximal w.r.t. inclusion are called *facets*. The *dimension* of a simplex $X \in S$ is $\dim(X) = |X| - 1$. The dimension of the simplicial complex $\mathcal{C}$ is $\dim(\mathcal{C}) = \sup\{\dim(X) \mid X \in S\}$. A simplicial complex is *pure* if all its facets have the same dimension (which is also the dimension of the complex). We say that a simplicial complex $\mathcal{C} = (V, S)$ is a *subcomplex* of $\mathcal{C}' = (V', S')$ when $S \subseteq S'$, and we write it $\mathcal{C} \subseteq \mathcal{C}'$.

Fix a finite set A whose elements are called *colors*. A *chromatic simplicial complex* $\mathcal{C} = (V, S, \chi)$ consists of a simplicial complex (V, S) equipped with a *coloring map* $\chi : V \to A$ such that every simplex $X \in S$ has vertices of different colors. A *chromatic simplicial map* $f : \mathcal{C} \to \mathcal{C}'$ from $\mathcal{C} = (V, S, \chi)$ to $\mathcal{C}' = (V', S', \chi')$ is a mapping $f : V \to V'$ between the vertices of the two complexes, such that:

- the image of a simplex is a simplex, i.e., for every $X \in S$, $f(X) := \{f(v) \mid v \in X\} \in S'$,
- f is *color-preserving*, i.e., for every $v \in V$, $\chi'(f(v)) = \chi(v)$.

A *chromatic carrier map* Φ from $\mathcal{C}$ to $\mathcal{C}'$, written $\Phi : \mathcal{C} \to 2^{\mathcal{C}'}$, assigns to each simplex $X \in S$ a subcomplex $\Phi(X)$ of $\mathcal{C}'$, such that:

- Φ is monotonic, i.e., if $Y \subseteq X$ then $\Phi(Y) \subseteq \Phi(X)$,
- Φ is rigid, i.e., for every simplex $X \in S$ of dimension d, $\Phi(X)$ is pure of dimension d,
- Φ is chromatic, i.e., for every $X \in S$, $\chi(X) = \chi'(\Phi(X))$, where $\chi(X) = \{\chi(v) \mid v \in X\}$ and $\chi'(\Phi(X)) = \bigcup_{Z \in \Phi(X)} \chi'(Z)$.

Given a chromatic carrier map $\Phi : \mathcal{C} \to 2^{\mathcal{C}'}$ and a chromatic simplicial map $f : \mathcal{C}' \to \mathcal{C}''$, their composition $f \circ \Phi : \mathcal{C} \to 2^{\mathcal{C}''}$ is defined as $(f \circ \Phi)(X) = \bigcup_{Z \in \Phi(X)} f(Z)$. Finally, given two chromatic carrier maps Φ and Ψ from $\mathcal{C}$ to $\mathcal{C}'$, we say that Φ is *carried by* Ψ, written $\Phi \subseteq \Psi$, when for every simplex $X \in \mathcal{C}$, $\Phi(X) \subseteq \Psi(X)$.

4.2 Simplicial tasks

The notion of task that we had in Definition 5 was a direct reformulation of the usual definition that is used in the context of combinatorial topology. We now recall the usual definition, and explain briefly why it is the same as Definition 5. Recall that the set of values is $\mathcal{V}$, and n is the number of processes. From now on, when we talk about chromatic complexes, the underlying set of colors will be $[n]$.

▶ **Definition 12.** *A* simplicial task *is a triple* $(\mathcal{I}, \mathcal{O}, \Xi)$, *where:*

- $\mathcal{I} = (V_\mathcal{I}, S_\mathcal{I}, \chi_\mathcal{I})$ *is the pure chromatic simplicial complex of dimension* $(n-1)$, *whose vertices are of the form* (i, v) *for all* $i \in [n]$ *and* $v \in \mathcal{V}$, *and which contains all the simplices that are well-colored.* $\mathcal{I}$ *is called the* input complex.
- $\mathcal{O} = (V_\mathcal{O}, S_\mathcal{O}, \chi_\mathcal{O}, \ell_\mathcal{O})$ *is a pure chromatic simplicial complex of dimension* $(n-1)$, *together with a labeling* $\ell_\mathcal{O} : V_\mathcal{O} \to \mathcal{V}$, *such that every vertex is uniquely identified by its color and its label.* $\mathcal{O}$ *is called the* output complex.
- $\Xi : \mathcal{I} \to 2^\mathcal{O}$ *is a chromatic carrier map from* $\mathcal{I}$ *to* $\mathcal{O}$.

As in Definition 5, we force the input complex to contain every possible combination of input values, which is unusual but can safely be assumed without loss of generality. There is a straightforward bijection between tasks and simplicial tasks. Consider a task $\Theta = (I, O, \Delta)$ in the sense of Definition 5. A vector $U \in (\mathcal{V} \cup \{\bot\})^n$ corresponds to the simplex $X = \{(i, U_i) \mid i \in [n] \text{ and } U_i \neq \bot\}$, where the vertex (i, U_i) is colored by i and labeled by U_i. Two matching vectors correspond to simplices with the same dimension and set of colors. A maximal vector (i.e., with no $\bot$ component) corresponds to a simplex of dimension $(n-1)$. Thus, the sets of maximal vectors I and O of a task correspond to the facets of the corresponding simplicial complexes $\mathcal{I}$ and $\mathcal{O}$; and their downward closures $\downarrow I$ and $\downarrow O$ correspond to $\mathcal{I}$ and $\mathcal{O}$. The fact that the relation $\Delta \subseteq \downarrow I \times \downarrow O$ relates only matching vectors, along with the non-emptiness of $\Delta(U)$, is equivalent to saying that the corresponding carrier map is rigid and chromatic. The last remaining condition is monotonicity of Δ, which must also hold for carrier maps. With that correspondence in mind, in the rest of the section, we will not distinguish tasks and simplicial tasks.

4.3 Simplicial protocols

The topological version of a protocol is defined similarly as for simplicial tasks:

▶ **Definition 13.** *A* simplicial protocol *is a triple* $(\mathcal{I}, \mathcal{P}, \Psi)$, *where:*

- $\mathcal{I} = (V_\mathcal{I}, S_\mathcal{I}, \chi_\mathcal{I})$ *is the pure chromatic simplicial complex of dimension* $(n-1)$, *whose vertices are of the form* (i, v) *for all* $i \in [n]$ *and* $v \in \mathcal{V}$, *and which contains all the simplices that are well-colored.* $\mathcal{I}$ *is called the* input complex.
- $\mathcal{P} = (V_\mathcal{P}, S_\mathcal{P}, \chi_\mathcal{P}, \ell_\mathcal{P})$ *is a pure chromatic simplicial complex of dimension* $(n-1)$, *together with a labeling* $\ell_\mathcal{P} : V_\mathcal{P} \to \mathsf{Views}$, *where* Views *is an arbitrary set of* views, *such that every vertex is uniquely identified by its color and its label.* $\mathcal{P}$ *is called the* protocol complex.
- $\Psi : \mathcal{I} \to 2^\mathcal{P}$ *is a chromatic carrier map from* $\mathcal{I}$ *to* $\mathcal{P}$.

The intended meaning of Definitions 12 and 13 is the following: the simplicial protocol $(\mathcal{I}, \mathcal{P}, \Psi)$ *solves* the simplicial task $(\mathcal{I}, \mathcal{O}, \Xi)$ if there exists a chromatic simplicial map $\delta : \mathcal{P} \to \mathcal{O}$ such that $\delta \circ \Psi$ is carried by Ξ. In [6], this is taken as the definition of what it means for a protocol to solve a task. In Section 2, we have given more concrete definitions of protocols and solvability; our goal here is to properly define the protocol complex associated to a given protocol, so that these two definitions of solvability will agree.

Let Obj be the set of objects that our programs are allowed to use. Let $P = (P_i)_{i\in[n]}$ be a wait-free protocol in the sense of Section 2.2. Recall that each $P_i = (Q_i, \bot_i, \delta_i, \tau_i)$ is the program of process i. As before, we write $\mathcal{A} = \{\mathsf{i}_i^x, \mathsf{r}_i^x, \mathsf{i}(o)_i^x, \mathsf{r}(o)_i^x \mid o \in \mathsf{Obj}, i \in [n], x \in \mathcal{V}\}$ for the set of actions of the protocol. The effect of a trace $T \in \mathcal{A}^*$ on global states is the function denoted by $\Delta : \overline{Q} \times \mathcal{A}^* \to \overline{Q}$.

The states $q \in Q_i \setminus \{\bot_i\}$ such that $\delta_i(q) \in \mathcal{V}$ are called the *final states* of process i. Let $F_i \subseteq Q_i$ denote the set of final states of process i. A trace $T \in \mathcal{A}^*$ is *one-shot* if it contains at most one outer invocation i_i^x and one outer response r_i^x for each process i. In particular, there is no restriction on the number of inner actions, since the objects in Obj are not assumed to be one-shot. A one-shot trace $T \in \mathcal{A}^*$ is *terminating* if after executing it, each process which participates in T ends in a final state. More formally, if we write $q = \Delta(q_{\text{init}}, T)$, for each i such that i_i^x occurs in T, we must have $q_i \in F_i$. Note that in a valid terminating trace, no action r_i^x occurs: when a process is in a final state, the process is ready to return its output value, but it did not return it yet.

We can now define the protocol complex $\mathcal{P} = (V_\mathcal{P}, S_\mathcal{P}, \chi_\mathcal{P}, \ell_\mathcal{P})$ associated to P. The set of views is $\mathsf{Views} = \bigcup_i F_i$. The vertices are of the form (i, q_i) where $i \in [n]$ is a process number and $q_i \in F_i$ is a final state of process i. Such a vertex is colored by i and labeled by q_i. Finally, for each one-shot trace T which is valid and terminating, we get a simplex $Y_T = \{(i, q_i) \mid i \text{ participates in } T\} \in S_\mathcal{P}$, where q_i is the i-th component of $\Delta(q_{\text{init}}, T)$. The carrier map $\Psi : \mathcal{I} \to 2^\mathcal{P}$ is defined as follows. For $X \in \mathcal{I}$ an input simplex, we let $S = \chi_\mathcal{I}(X)$ be the set of participating processes, and $(v_i)_{i\in S}$ their input values. Then, $\Psi(X)$ consists of all simplexes Y_T where T is a valid terminating one-shot trace such that every invocation that occurs in T is of the form $\mathsf{i}_i^{v_i}$ for some $i \in S$.

It is straightforward to see that $\mathcal{P}$ is chromatic, and that Ψ is monotonic and chromatic. To check that $\mathcal{P}$ is pure of dimension $(n-1)$, let Y_T be a simplex of $\mathcal{P}$. We want to extend it to an $(n-1)$-dimensional simplex. To do so, first we append at the end of the trace T invocations i_i^x for each process i that does not participate in T. Since we assumed that the protocol P is wait-free, we can run each of these processes until it reaches a final state. Thus, we get a trace T' which is valid and terminating, and the corresponding simplex $Y_{T'}$ is of dimension $(n-1)$ and contains Y_T. Checking that Ψ is rigid is similar.

4.4 Asynchronous computability theorem

Let $\Theta = (\mathcal{I}, \mathcal{O}, \Xi)$ be a task, $P = (P_i)_{i\in[n]}$ a protocol, and $(\mathcal{I}, \mathcal{P}, \Psi)$ its associated simplicial protocol as described in the previous section. Notice that, since we defined the vertices of $\mathcal{P}$ to be pairs (i, q_i) where $q_i \in F_i$ is a final state of process i, there is a map $\delta : V_\mathcal{P} \to \mathcal{V}$ defined as $\delta(i, q_i) = \delta_i(q_i)$, where δ_i is the decision function of the program P_i.

The following Theorem gives a topological characterization of what it means for the protocol P to implement (in the sense of Definition 4) the task Θ:

▶ **Theorem 14.** *The protocol P implements the task-object $G(\Theta)$ if and only if the map $\delta : V_\mathcal{P} \to \mathcal{V}$ induces a chromatic simplicial map $\delta : \mathcal{P} \to \mathcal{O}$ such that $\delta \circ \Psi$ is carried by Ξ.*

Proof. In this proof, to distinguish between the execution traces of P (on the alphabet $\{\mathsf{i}_i^x, \mathsf{r}_i^x, \mathsf{i}(o)_i^x, \mathsf{r}(o)_i^x\}$) and the outer traces (on the alphabet $\{\mathsf{i}_i^x, \mathsf{r}_i^x\}$), we use lowercase letters t, t', s, s' for the former and capital letters T, T', S, S' for the latter.

($\Rightarrow$): Assume that P implements the object $G(\Theta)$, i.e., every trace $T \in [\![P]\!]$ satisfies the conditions of Definition 6. First, we want to define the map $\delta : V_\mathcal{P} \to V_\mathcal{O}$. Let $(i, q_i) \in V_\mathcal{P}$ be a vertex of $\mathcal{P}$; we would like to take $\delta(i, q_i) = (i, \delta_i(q_i))$, but we do not know yet that it is a vertex of $\mathcal{O}$. The vertex (i, q_i) belongs to a $(n-1)$-dimensional simplex Y_t for some execution trace t which is valid, one-shot and terminating. Let $q = \Delta(q_{\text{init}}, t)$ be the global state after executing t. In particular, the i-th component of q is q_i. For each $j \in [n]$, write $d_j = \delta_j(q_j)$ the value that process j is about to decide in the trace t. We also write $v_j \in \mathcal{V}$ the input value of process j in t. Let t' denote the trace obtained by appending responses $\mathsf{r}_j^{d_j}$ at the end of t. Then t' is still a valid trace, so if we write $T = \pi(t)$ its projection, we get $T \in [\![P]\!] \subseteq G(\Theta)$. Since T is a complete trace, that means it respects the task specification, i.e., $\{(j, d_j) \mid j \in [n]\} \in \Xi(\{(j, v_j) \mid j \in [n]\})$. In particular, (i, d_i) is in the image of Ξ, so it is a vertex of $\mathcal{O}$.

The map $\delta : V_{\mathcal{P}} \to V_{\mathcal{O}}$ is obviously chromatic. Let us show that it is a simplicial map. Let $Y_t \in S_{\mathcal{P}}$ be a simplex of $\mathcal{P}$. Let $S \subseteq [n]$ be the set of processes participating in t, then Y_t is of the form $Y_t = \{(i, q_i) \mid i \in S\}$, and $\delta(Y_t) = \{(i, d_i) \mid i \in S\}$. As we did in the previous paragraph, we can add responses $\mathsf{r}_i^{d_i}$ for $i \in S$ at the end of the trace t, to obtain a complete valid trace whose projection is in $[\![P]\!]$, and thus also in $G(\Theta)$. This implies that $\delta(Y_t)$ is in the image of Ξ, so it is a simplex of $\mathcal{O}$.

Finally, we need to show that $\delta \circ \Psi$ is carried by Ξ. Let $X \in \mathcal{I}$ be an input simplex, and let $Z \in (\delta \circ \Psi)(X)$ be an output simplex. Then Z must be of the form $\delta(Y_t)$ for some $Y_t \in \Psi(X)$. We write $S \subseteq [n]$ the set of participating processes of X, and $(v_i)_{i \in S}$ their input values. So, t is a valid terminating one-shot trace such that every invocation in t is of the form $\mathsf{i}_i^{v_i}$ for some $i \in S$. Let $S' \subseteq S$ be the participating set of t. Let $q = \Delta(q_{\text{init}}, t)$ be the global state after executing t; we have $Y_t = \{(i, q_i) \mid i \in S'\}$, and $\delta(Y_t) = \{(i, d_i) \mid i \in S'\}$. Once again, we can append appropriate responses $\mathsf{r}_i^{d_i}$ to the trace t, and then we obtain $\pi(t') = T \in [\![P]\!] \subseteq G(\Theta)$. So, we obtain $\delta(Y_t) \in \Xi(\{(i, v_i) \mid i \in S'\})$. Since $\{(i, v_i) \mid i \in S'\} \subseteq X$, by monotonicity of Ξ, we finally get $\delta(Y_t) \in \Xi(X)$.

($\Leftarrow$): Assume that the induced map $\delta : V_{\mathcal{P}} \to V_{\mathcal{O}}$ defined as $\delta(i, q_i) = \delta_i(q_i)$ is a chromatic simplicial map from $\mathcal{P}$ to $\mathcal{O}$, and that $\delta \circ \Psi$ is carried by Ξ. We want to prove that P implements the object $G(\Theta)$, i.e., that $[\![P]\!] \subseteq G(\Theta)$. Let $T \in [\![P]\!]$ be a one-shot outer trace, and t such that $\pi(t) = T$ a valid execution trace for P. To show that $T \in G(\Theta)$, we must first complete it, that is, find valid responses to the pending invocations of T. Since the protocol P is wait-free, we can just run the pending processes one by one until they all reach a final state: formally, this amounts to appending inner actions to the trace t according to its program, until a final state q_i is reached. Then, we add the appropriate response $\mathsf{r}_i^{\delta_i(q_i)}$ to obtain a trace t', which extends t, is still valid, and which does not have pending invocations. The projection $T' = \pi(t') \in [\![P]\!]$ is an extension of T where we added responses to the pending invocations of T.

Now that we have T', we have to prove the following property on every non-empty prefix S of T' which is complete (including $S = T'$): the simplex Z_S must belong to $\Xi(X_S)$, where X_S is the input simplex $X_S = \{(i, v_i) \mid \mathsf{i}_i^{v_i} \text{ occurs in } S\}$ and Z_S is the output simplex $Z_S = \{(i, d_i) \mid \mathsf{r}_i^{d_i} \text{ occurs in } S\}$. Our goal is now to decompose Z_S as $Z_S = \delta(Y)$ for some $Y \in \Psi(X_S)$. Let s be the prefix of t' whose projection is $\pi(s) = S$. We write s' for the trace obtained by removing from s all the outer responses, i.e., actions of the form $\mathsf{r}_i^{d_i}$. We claim that s' is still a valid trace: indeed, no action from process i can occur after the outer response (otherwise s would not be one-shot), and the only effect of $\mathsf{r}_i^{d_i}$ is to change the local state of i, which does not affect the validity of the actions of other processes. Moreover, in the global states $q = \Delta(q_{\text{init}}, s')$, every process that participates in s' is in a final state, in other words, s' is terminating. Thus, we have a simplex $Y_{s'} \in S_{\mathcal{P}}$ in the protocol complex consisting of all the vertices (i, q_i) where i participates in s'. Since s', s and S all have the same participating set, $Y_{s'} \in \Psi(X_S)$. And since $d_i = \delta_i(q_i)$ (because in s', process i is ready to decide $\mathsf{r}_i^{d_i}$), $\delta(Y_{s'}) = Z_S$.

This decomposition of Z_S shows that $Z_S \in \delta \circ \Psi(X_S)$. Since we assumed that $\delta \circ \Psi$ is carried by Ξ, this implies $Z_S \in \Xi(X_S)$, which concludes the proof. ◀

Despite the verbosity of the proof, nothing complicated is going on: we are just putting together all the definitions of the paper. In particular, the crucial definitions that allow the proof to go through are the expansion property (5) in Definition 1; the map G (Definition 6) that characterizes the objects which correspond to tasks; and the definition of the protocol complex $\mathcal{P}$ associated to a protocol P in Section 4.3.

If we instantiate Theorem 14 with Obj containing only an iterated immediate snapshot object, combined with the fact that immediate snapshot protocol complexes are subdivisions of the input complex, we obtain Herlihy and Shavit's asynchronous computability theorem for read/write registers. In general, if we fix a particular set of objects Obj, to prove that a task cannot be solved using the objects of Obj, one needs to find a topological invariant which holds in *any* protocol complex $\mathcal{P}$ associated to any protocol P. This is usually where all the difficulty of the proof lies, and of course Theorem 14 does not help with that part. What the Theorem says is merely that solvability in the sense of protocol complexes agrees with the more basic notion of solvability defined in Section 4.

5 Conclusion

We have extended Herlihy and Shavit's asynchronous computability theorem, which gives a topological characterization of the solvability of tasks, not only in the context of read/write registers, but for a large class of arbitrary objects. This shows the soundness of the topological approach to fault-tolerant computability, as described for example in [6], where the existence of a simplicial map from the protocol complex to the output complex is taken as the *definition* of task solvability. A practical benefit of our proof is that we gave a general definition of what the protocol complex should be to model arbitrary objects. One can now instantiate this definition with a given protocol, without risk of making a mistake. This work paves the way towards a better understanding of the protocol complex: rather than seeing it as a huge, monolithic combinatorial object, we would like to construct it in a modular way, by decomposing it into more basic components.

To establish this theorem, we had to study the distinction between tasks and one-shot objects. We characterized the class of one-shot objects which correspond to tasks, and this result might be of independent interest. A natural continuation of this work would be to try to characterize the solvability of all one-shot objects, possibly using the notion of *refined task* introduced in [2]. Another generalization would be to extend it to t-resilient solvability, instead of focusing on wait-free protocols.

References

1 Elizabeth Borowsky and Eli Gafni. Generalized FLP Impossibility Result for T-resilient Asynchronous Computations. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 91–100, New York, NY, USA, 1993. ACM. doi:10.1145/167088.167119.

2 Armando Castañeda, Sergio Rajsbaum, and Michel Raynal. Unifying Concurrent Objects and Distributed Tasks: Interval-Linearizability. *J. ACM*, 65(6):45:1–45:42, 2018. doi:10.1145/3266457.

3 Ivana Filipović, Peter O'Hearn, Noam Rinetzky, and Hongseok Yang. Abstraction for concurrent objects. *Theoretical Computer Science*, 411(51):4379–4398, 2010. European Symposium on Programming 2009.

4 Eli Gafni, Sergio Rajsbaum, and Maurice Herlihy. Subconsensus Tasks: Renaming Is Weaker Than Set Agreement. In Shlomi Dolev, editor, *Distributed Computing*, pages 329–338. Springer Berlin Heidelberg, 2006.

5 Éric Goubault, Jérémy Ledent, and Samuel Mimram. Concurrent Specifications Beyond Linearizability. In *22nd International Conference on Principles of Distributed Systems, OPODIS 2018*, pages 28:1–28:16, 2018. doi:10.4230/LIPIcs.OPODIS.2018.28.

6 Maurice Herlihy, Dmitry Kozlov, and Sergio Rajsbaum. *Distributed Computing Through Combinatorial Topology.* Morgan Kaufmann Publishers Inc., 2013.

7 Maurice Herlihy and Sergio Rajsbaum. Set Consensus Using Arbitrary Objects (Preliminary Version). In *Proceedings of the Thirteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '94, pages 324–333, New York, NY, USA, 1994. ACM. doi:10.1145/197917.198119.

8 Maurice Herlihy and Sergio Rajsbaum. *Algebraic topology and distributed computing: a primer*, pages 203–217. Springer Berlin Heidelberg, 1995. doi:10.1007/BFb0015245.

9 Maurice Herlihy, Sergio Rajsbaum, and Mark R. Tuttle. Unifying Synchronous and Asynchronous Message-passing Models. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '98, pages 133–142, New York, NY, USA, 1998. ACM. doi:10.1145/277697.277722.

10 Maurice Herlihy and Nir Shavit. The Topological Structure of Asynchronous Computability. *J. ACM*, 46(6):858–923, November 1999. doi:10.1145/331524.331529.

11 Maurice Herlihy and Jeannette M. Wing. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, 1990.

12 Nancy A. Lynch and Mark R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2:219–246, 1989. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.83.7751.

13 Michael Saks and Fotios Zaharoglou. Wait-free K-set Agreement is Impossible: The Topology of Public Knowledge. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 101–110, New York, NY, USA, 1993. ACM. doi:10.1145/167088.167122.

Game-Based Local Model Checking for the Coalgebraic μ-Calculus

Daniel Hausmann
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
daniel.hausmann@fau.de

Lutz Schröder
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany
lutz.schroeder@fau.de

Abstract

The coalgebraic μ-calculus is a generic framework for fixpoint logics with varying branching types that subsumes, besides the standard relational μ-calculus, such diverse logics as the graded μ-calculus, the monotone μ-calculus, the probabilistic μ-calculus, and the alternating-time μ-calculus. In the present work, we give a local model checking algorithm for the coalgebraic μ-calculus using a coalgebraic variant of parity games that runs, under mild assumptions on the complexity of the so-called one-step satisfaction problem, in time p^k where p is a polynomial in the formula and model size and where k is the alternation depth of the formula. We show moreover that under the same assumptions, the model checking problem is in $\mathrm{NP} \cap \mathrm{coNP}$, improving the complexity in all mentioned non-relational cases. If one-step satisfaction can be solved by means of small finite games, we moreover obtain standard parity games, ensuring quasi-polynomial run time. This applies in particular to the monotone μ-calculus, the alternating-time μ-calculus, and the graded μ-calculus with grades coded in unary.

2012 ACM Subject Classification Theory of computation → Modal and temporal logics; Theory of computation → Verification by model checking

Keywords and phrases Model checking, μ-calculus, coalgebraic logic, graded μ-calculus, probabilistic μ-calculus, parity games

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.35

Funding Work forms part of the DFG project *Generic Algorithmic Methods for Modal and Hybrid Logics* (SCHR 1118/5-3).

1 Introduction

One of the most central and established computation tasks in the verification of concurrent software is *model checking*, i.e. to determine whether a given system state satisfies a temporal specification (e.g. [2]). The complexity of model checking is generally fairly sensitive to variations in the logic, more so than satisfiability checking, which for fairly wide ranges of branching time temporal logics (such as PDL, CTL and the μ-calculus) tends to be ExpTime-complete. For instance, CTL allows model checking in polynomial time, while LTL (and more generally CTL*) model checking is PSpace-complete (e.g. [27]). In fact, the input of model checking is naturally split into two parts, the model and the formula, and the complexity analysis is typically phrased in terms of model size and formula size separately.

Model checking for μ-calculus formulae typically proceeds by a reduction to computing winning regions in parity games, and in fact the two problems are linear-time equivalent [11]. In consequence, the current best upper bound for the time complexity of μ-calculus model checking has recently dropped when Calude et al. [4] showed that parity games can be solved in quasi-polynomial time; in fact, further improvement seems possible as the exact complexity of parity game solving remains open.

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 35; pp. 35:1–35:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Besides model checking relational systems, there is a long-standing and growing interest in systems with additional structure, e.g. probabilities, weights, multi-player games, or neighbourhoods. To name just a few concrete examples, the alternating-time μ-calculus [1] is interpreted over concurrent game structures; the (two-valued) probabilistic μ-calculus [6, 21] over Markov chains, and the monotone μ-calculus [12] (the ambient fixpoint logic of concurrent dynamic logic (CPDL) [25] and Parikh's game logic [23]) over (monotone) neighbourhood structures. The graded μ-calculus [20], standardly interpreted over relational structures, has an equivalent and more natural semantics over integer-weighted structures [10]. As a unifying framework for such fixpoint logics beyond relational semantics, the *coalgebraic μ-calculus* [6] has emerged. It works within the paradigm of *universal coalgebra* [26], i.e. encapsulates the system type as a set functor and systems as coalgebras for this functor; the interpretation of modalities is based on *predicate liftings* as used in the broader field of *coalgebraic logic* [8].

Our present contribution is a generic *local* model checking algorithm for the coalgebraic μ-calculus, which can be instantiated easily to concrete μ-calculi including all the ones mentioned above, where by *local* we mean that the algorithm allows establishing satisfaction of a formula in a given state without calculating full extensions of all subformulae across all states. Under mild assumptions on the concrete functor and predicate liftings defining the logic, our algorithm runs in time p^k where p is a polynomial in the size of the formula and the model and k is the alternation depth of the formula; in particular, the algorithm runs in polynomial time on alternation-free coalgebraic μ-calculi, or more generally on formulae of bounded alternation depth. Further analysis shows moreover that under the same assumptions, the model checking problem is in fact in NP $\cap$ coNP. The algorithm is based on a coalgebraic generalization of parity games, in which some steps consist in calls to a *one-step satisfaction* checker that essentially just implements the modalities. In some cases, notably the monotone μ-calculus, the alternating-time μ-calculus, and the graded μ-calculus, we can replace one-step satisfaction checking with suitable finite *one-step satisfaction games*, obtaining a standard (rather than coalgebraic) parity game. Exploiting the mentioned recent results on parity game solving, we obtain quasi-polynomial runtime in cases where these one-step satisfaction games are sufficiently small; in particular, we show that the monotone μ-calculus, the alternating-time μ-calculus, and the graded μ-calculus with unary coding of grades all admit model checking in quasi-polynomial time, to our knowledge new results.

Related Work. Model checking games for the monotone μ-calculus have been studied by Hansen et al. [15], without complexity analysis. For the alternating-time μ-calculus, Alur et al. [1] describe a model checking procedure that works essentially by fixpoint iteration, and runs in time $\mathcal{O}((m \cdot l)^{k+1})$ where m is the number of transitions, l the length of the formula, and k its alternation depth. An upper bound UP $\cap$ coUP for model checking the probabilistic μ-calculus is stated in [21]. The complexity of model checking the graded μ-calculus is mentioned by Ferrante et al. [13] in work otherwise concerned with the more complex problem of *module* checking, giving an upper bound ExpTime for numbers in both systems and formulae coded in binary. We improve this bound to NP $\cap$ coNP under binary coding, and to quasi-polynomial time under unary coding, and moreover give a deterministic algorithm that is exponential only in the alternation depth.

Cîrstea et al. [9] provide an automata-based model checking procedure for quantitative linear time logics over systems with weights in a partial commutative semiring, using coalgebraic methods. Satisfiability checking in the coalgebraic μ-calculus has been shown to be in ExpTime [6, 17]. For purposes of the present paper, the most relevant piece of related work is Hasuo et al.'s [16] model checking algorithm for the coalgebraic μ-calculus, which is based on fixpoint computation using progress measures in a highly general setting. The bound on the run time stated in [16] has roughly the form p^r where p is a polynomial in the number of states of the model and the size of the formula understood as the number of nodes in its parse tree, and r is the number of least fixpoint operators. The bound is thus independent of the size of the transition structure of models and the actual representation size of the formula (which may in general contain, e.g., integer or rational numbers); this can clearly hold only under strong assumptions on both the system type and the syntax of formulae, mentioned implicitly in [16] on p. 728. In particular, the overall representation size of models must be polynomially bounded in the number of states. The analyis in [16] therefore does not apply to our main examples; e.g. a concurrent game structure with just one state can have an unbounded number of transitions; integer-weighted transition systems can involve unboundedly large integer numbers; and a monotone neighbourhood frame can have exponentially more (even minimal) neighbourhoods than states. Another important difference is that we make the intermediate game theoretic constructions in the algorithm explicit, while game constructions are eliminated in favour of a direct implementation of progress measures in [16]. Immediate benefits include exponential dependence only on alternation depth instead of number of least fixpoint operators, materializing an improvement conjectured by Hasuo et al.; the local nature of our model checking algorithm; and quasi-polynomial runtime for the above-mentioned cases admitting small one-step satisfaction games. As a long-term benefit, we expect algorithmic improvements paralleling the development in standard parity games also for our coalgebraic parity games, and hence for the complexity of coalgebraic μ-calculus model checking in general.

2 The Coalgebraic μ-Calculus

We proceed to recall basic definitions and examples in universal coalgebra [26] and the coalgebraic μ-calculus [6].

The abstraction principle underlying universal coalgebra is to encapsulate system types as functors, for our present purposes on the category of sets. Such a functor $T : \mathsf{Set} \to \mathsf{Set}$ maps every set X to a set $T(X)$, and every map $f : X \to Y$ to a map $Tf : T(X) \to T(Y)$, preserving identities and composition. We think of $T(X)$ as a type of structured collections over X; a basic example is the covariant powerset functor $\mathcal{P}$, which assigns to each set its powerset and acts on maps by taking forward image. Systems of the intended type are then cast as *T-coalgebras* (C, ξ) (or just ξ) consisting of a set C of *states* and a *transition map* $\xi : C \to T(C)$, thought of as assigning to each state $x \in C$ a structured collection $\xi(x) \in T(C)$ of successors. E.g. a $\mathcal{P}$-coalgebra $\xi : C \to \mathcal{P}(C)$ assigns to each state a set of successors; that is, $\mathcal{P}$-coalgebras are transition systems. We will see additional examples later. A coalgebra is *finite* if its state set is finite.

Following the paradigm of *coalgebraic logic* [8], we fix a set Λ of modal operators (in principle of any finite arity but restricted to unary modalities in the technical development for the sake of readability; our proofs generalize by essentially writing more indices), which we interpret over T-coalgebras for a functor T as *predicate liftings*, i.e. natural transformations

$$[\![\heartsuit]\!]_X : 2^X \to 2^{T(X)} \qquad \text{for } \heartsuit \in \Lambda.$$

Here, the index X, omitted when clear from the context, ranges over all sets; 2^X denotes the set of maps $X \to 2$ into the two-element set $2 = \{\bot, \top\}$, isomorphic to the powerset of X (i.e. 2^- is the *contravariant powerset functor*); and naturality means that $[\![\heartsuit]\!]_X(f^{-1}[A]) =$

$(Tf)^{-1}[[\![\heartsuit]\!]_Y(A)]$ for $f : X \to Y$ and $A \in 2^Y$. Thus, the predicate lifing $[\![\heartsuit]\!]$ indeed lifts predicates on a base set X to predicates on the set $T(X)$. Two standard examples for $T = \mathcal{P}$ are the predicate liftings for the standard $\Box$ and $\Diamond$ modalities, given by

$$[\![\Box]\!]_X(A) = \{B \in \mathcal{P}(X) \mid B \subseteq A\} \quad \text{and} \quad [\![\Diamond]\!]_X(A) = \{B \in \mathcal{P}(X) \mid A \cap B \neq \emptyset\}.$$

Since we mean to form fixpoint logics, we need to require that every $[\![\heartsuit]\!]$ is *monotone*, that is, $A \subseteq B \subseteq X$ implies $[\![\heartsuit]\!]_X(A) \subseteq [\![\heartsuit]\!]_X(B)$. To support negation, we assume moreover that Λ is closed under *duals*, i.e. for each $\heartsuit \in \Lambda$ we have $\overline{\heartsuit} \in \Lambda$ such that $[\![\overline{\heartsuit}]\!]_X(A) = T(X) \setminus [\![\heartsuit]\!]_X(X \setminus A)$, chosen so that $\overline{\overline{\heartsuit}} = \heartsuit$ (e.g. $\overline{\Box} = \Diamond$, $\overline{\Diamond} = \Box$).

To introduce the syntax of the *coalgebraic μ-calculus*, we fix a set Var of *fixpoint variables*. We let the meta-variable η range over the standard fixpoint operators μ (least fixpoint), ν (greatest fixpoint). The set of *formulae* ϕ, ψ is then defined by the grammar

$$\psi, \phi := \top \mid \bot \mid \psi \vee \phi \mid \psi \wedge \phi \mid \heartsuit\psi \mid X \mid \eta X.\psi \qquad (\heartsuit \in \Lambda, X \in \mathsf{Var}).$$

Note that the grammar does not include propositional atoms; these can, if desired, be treated as nullary modalities (Example 1). Negation is not included explicitly but can be defined, as usual, by taking negation normal forms. Fixpoint operators bind their variables, giving rise to the usual notions of bound and free variables; we denote the set of free variables of a formula ψ by $\mathsf{FV}(\psi)$. A formula ψ is *closed* if $\mathsf{FV}(\psi) = \emptyset$. Given a T-coalgebra $\xi : C \to T(C)$ and a valuation $\sigma : \mathsf{Var} \to \mathcal{P}(C)$, the extension

$$[\![\phi]\!]_\sigma \subseteq C$$

of a formula ϕ is defined recursively by the expected clauses for the propositional operators ($[\![\top]\!]_\sigma = C$; $[\![\bot]\!]_\sigma = \emptyset$; $[\![\phi \wedge \psi]\!]_\sigma = [\![\phi]\!]_\sigma \cap [\![\psi]\!]_\sigma$; $[\![\phi \vee \psi]\!]_\sigma = [\![\phi]\!]_\sigma \cup [\![\psi]\!]_\sigma$); by $[\![X]\!]_\sigma = \sigma(X)$ and

$$[\![\heartsuit\psi]\!]_\sigma = \xi^{-1}[[\![\heartsuit]\!]([\![\psi]\!]_\sigma)];$$

and by $[\![\mu X.\psi]\!]_\sigma = \mathsf{LFP}[\![\psi]\!]^X_\sigma$, $[\![\nu X.\psi]\!]_\sigma = \mathsf{GFP}[\![\psi]\!]^X_\sigma$ where the map $[\![\psi]\!]^X_\sigma : \mathcal{P}(C) \to \mathcal{P}(C)$ is defined by $[\![\psi]\!]^X_\sigma(A) = [\![\psi]\!]_{\sigma[X \mapsto A]}$ for $A \subseteq C$, with $(\sigma[X \mapsto A])(X) = A$ and $(\sigma[X \mapsto A])(Y) = \sigma(Y)$ for $X \neq Y$, and LFP and GFP take least and greatest fixpoints of monotone functions on $\mathcal{P}(C)$, respectively; monotonicity of $[\![\psi]\!]^X_\sigma$ is clearly an invariant of the recursive definition.

The *alternation depth* $\mathsf{ad}(\eta X.\psi)$ of a fixpoint $\eta X.\psi$ is the depth of alternating nesting of such fixpoints in ψ that depend on X; we assign *odd* numbers to least fixpoints and *even* numbers to greatest fixpoints. E.g. for $\psi = \nu X.\phi$ and $\phi = \mu Y.(p \wedge \heartsuit X) \vee \heartsuit Y$, we have $\mathsf{ad}(\psi) = 2$, $\mathsf{ad}(\phi) = 1$. For a detailed definition of alternation depth, see e.g. [22].

▶ **Example 1.** We proceed to see some standard examples of coalgebraic semantics, see [28, 6, 29] for more details.

1. *Relational μ-calculus:* As indicated above, we obtain a version of the standard relational μ-calculus [19] without propositional atoms by taking $T = \mathcal{P}$ and $\Lambda = \{\Box, \Diamond\}$ with predicate liftings as described previously. We can add a set At of propositional atoms $p, q, \ldots$ by regarding them as nullary modalities, and interpret $p \in \mathsf{At}$ over the extended functor $T = \mathcal{P}(\mathsf{At}) \times \mathcal{P}$ by the nullary predicate lifting $[\![p]\!]_X = \{(P, B) \in \mathcal{P}(\mathsf{At}) \times \mathcal{P}(X) \mid p \in P\}$. E.g. the formula $\nu X.\, \mu Y.\, ((p \wedge \Box X) \vee (q \wedge \Box Y))$ says that on every path from the current state, p holds everywhere except possibly on finite segments where q holds. Similarly, we can introduce a set Act of actions, and index modalites over actions $a \in \mathsf{Act}$. We interpret the indexed modalities $\Box_a, \Diamond_a$ over the functor $T = \mathcal{P}^{\mathsf{Act}}$, e.g. the box by $[\![\Box_a]\!]_X(A) = \{f \in \mathcal{P}(X)^{\mathsf{Act}} \mid f(a) \subseteq A\}$. We can similarly add propositional atoms and actions to all examples that follow.

2. *Graded μ-calculus:* The graded μ-calculus [20] has modalities $\langle b\rangle$, $[b]$, indexed over $b \in \mathbb{N}$, read "in more than b successors" and "in all but at most b successors", respectively. These can be interpreted over relational structures but it is technically more convenient to use *multigraphs*, i.e. transition systems with edge weights (*multiplicities*) in $\mathbb{N} \cup \{\infty\}$, which are coalgebras for the multiset functor $\mathcal{B}$. The latter maps a set X to the set $\mathcal{B}(X) = (\mathbb{N} \cup \{\infty\})^X$ of maps $X \to (\mathbb{N} \cup \{\infty\})$; we treat elements $\beta \in \mathcal{B}(X)$ as $(\mathbb{N} \cup \{\infty\})$-valued discrete measures on X, and in particular write $\beta(A) = \sum_{x\in A} \beta(x)$ for $A \subseteq X$. For a map $f : X \to Y$, the map $\mathcal{B}(f) : \mathcal{B}(X) \to \mathcal{B}(Y)$ is then given by $\mathcal{B}(f)(\beta)(y) = \beta(f^{-1}[\{y\}])$. Over $\mathcal{B}$-coalgebras, we interpret $\langle b\rangle$ and $[b]$ by the mutually dual predicate liftings

$$[\![\langle b\rangle]\!]_X(A) = \{\beta \in \mathcal{B}(X) \mid \beta(A) > b\} \quad \text{and} \quad [\![[b]]\!]_X(A) = \{\beta \in \mathcal{B}(X) \mid \beta(X \setminus A) \leq b\}.$$

E.g. the formula $\nu X.\, (\phi \wedge \Diamond_1 X)$ says that the current state is the root of an infinite tree with branching degree at least 2 (counting multiplicities) on which ϕ holds everywhere.

3. *Probabilistic μ-calculus:* Let $\mathcal{D}$ denote the *discrete distribution functor*, defined on sets by $\mathcal{D}(X) = \{\beta : X \to [0, 1] \mid \sum_{x\in X} \beta(x) = 1\}$. That is, $\mathcal{D}(X)$ is the set of discrete probability distributions on X; coalgebras for $\mathcal{D}$ are just Markov chains. Similarly as for the graded μ-calculus, we take modalities $[p]$, $\langle p\rangle$ indexed over $p \in [0, 1] \cap \mathbb{Q}$, interpreted over $\mathcal{D}$ by $[\![\langle p\rangle]\!]_X(A) = \{\beta \in \mathcal{D}(X) \mid \beta(A) > p\}$ and $[\![[p]]\!]_X(A) = \{\beta \in \mathcal{D}(X) \mid \beta(X \setminus A) \leq p\}$ (using the same measure-theoretic notation as in the previous item). The arising coalgebraic μ-calculus is the *probabilistic μ-calculus* [6, 21].

4. *Monotone μ-calculus:* The *monotone neighbourhood functor* $\mathcal{M}$ maps a set X to the set $\mathcal{M}(X) = \{\mathfrak{A} \in 2^{(2^X)} \mid \mathfrak{A} \text{ upwards closed}\}$ of set systems over X that are upwards closed under subset inclusion (i.e. $A \in \mathfrak{A}$ and $A \subseteq B$ imply $B \in \mathfrak{A}$). Coalgebras for $\mathcal{M}$ are *monotone neighbourhood frames* in the sense of Scott-Montague semantics [5]. We take $\Lambda = \{\Box, \Diamond\}$ and interpret $\Box$ over $\mathcal{M}$ by the predicate lifting

$$[\![\Box]\!]_X(A) = \{\mathfrak{A} \in \mathcal{M}(X) \mid A \in \mathfrak{A}\} = \{\mathfrak{A} \in \mathcal{M}(X) \mid \exists B \in \mathfrak{A}.\, B \subseteq A\},$$

and $\Diamond$ by the corresponding dual lifting, $[\![\Diamond]\!]_X(A) = \{\mathfrak{A} \in \mathcal{M}(X) \mid (X \setminus A) \notin \mathfrak{A}\} = \{\mathfrak{A} \in \mathcal{M}(X) \mid \forall B \in \mathfrak{A}.\, B \cap A \neq \emptyset\}$. The arising coalgebraic μ-calculus is known as the *monotone μ-calculus* [12]. When we add propositional atoms and actions, and replace $\mathcal{M}$ with its subfunctor $\mathcal{M}_s$ defined by $\mathcal{M}_s(X) = \{\mathfrak{A} \in \mathcal{M}(X) \mid \emptyset \notin \mathfrak{A} \ni X\}$, whose coalgebras are *serial* monotone neighbourhood frames, we arrive at the ambient fixpoint logic of *concurrent dynamic logic* [25] and Parikh's *game logic* [23]. In game logic, actions are understood as atomic games of Angel vs. Demon, and we read $\Box_a \phi$ as "Angel has a strategy to enforce ϕ in game a". Game logic is then mainly concerned with composite games, formed by the control operators of dynamic logic and additional ones; the semantics can be encoded into fixpoint definitions. For instance, the formula $\nu X.\, p \wedge \Box_a X$ says that Angel can enforce p in the composite game where a is played repeatedly, with Demon deciding when to stop.

5. *Alternating-time μ-calculus:* Fix a set $N = \{1, \dots, n\}$ of *agents*. Using alternative notation from *coalition logic* [24], we present the *alternating-time μ-calculus (AMC)* [1] by modalities $[D]$, $\langle D\rangle$ indexed over *coalitions* $D \subseteq N$, read "D can enforce" and "D cannot prevent", respectively. We define a functor $\mathcal{G}$ by

$$\mathcal{G}(X) = \{(k_1, \dots, k_n, f) \mid k_1, \dots, k_n \in \mathbb{N} \setminus \{0\}, f : \left(\prod_{i\in N}[k_i]\right) \to X\}$$

where we write $[k] = \{1, \dots, k\}$. We understand $(k_1, \dots, k_n, f) \in \mathcal{G}(X)$ as a one-step concurrent game with k_i available moves for agent $i \in N$, and outcomes in X determined by the *outcome function* f from a joint choice of moves by all the agents. For $D \subseteq N$, we write $S_D = \prod_{i \in D}[k_i]$. Given joint choices $s_D \in S_D$, $s_{N \setminus D} \in S_{N \setminus D}$ of moves for D and $N \setminus D$ respectively, we write $(s_D, s_{N \setminus D}) \in s_N$ for the joint move of all agents induced in the evident way. In this notation, we interpret the modalities $[D]$ over $\mathcal{G}$ by the predicate lifting

$$[\![[D]]\!]_X(A) = \{(k_1, \dots, k_n, f) \in \mathcal{G}(X) \mid \exists s_D \in S_D.\, \forall s_{N \setminus D} \in S_{N \setminus D}.\, f(s_D, s_{N \setminus D}) \in A\},$$

and the modalities $\langle D \rangle$ by dualization. This captures exactly the semantics of the AMC: $\mathcal{G}$-coalgebras are precisely *concurrent game structures* [1], i.e. assign a one-step concurrent game to each state, and $[D]\phi$ says that the agents in D have a joint move such that whatever the agents in $N \setminus D$ do, the next state will satisfy ϕ. E.g. $\mu X.\, p \vee [D]X$ says that coalition D can eventually enforce that p is satisfied (a property expressible already in alternating-time temporal logic ATL [1]).

We *fix the data T, Λ and a predicate lifting $[\![\heartsuit]\!]$ for each $\heartsuit \in \Lambda$ for the rest of the paper*. We assume given a suitable representation size for the modalities in Λ; this is relevant in particular when Λ is infinite, e.g. for the graded and the probabilistic μ-calculus. We generally assume that numbers are coded in binary, except in the treatment of the graded μ-calculus via one-step satisfaction games in Section 5.

We write $\mathsf{size}(\psi)$ for the ensuing representation size of a formula ψ, counting the representation size for each modality and 1 for other connectives and variables. Similarly, we assume given a representation of elements of $T(X)$ for finite sets X, with associated representation size $\mathsf{size}(s)$ for elements $s \in T(X)$. Again, we generally assume that numbers in s (e.g. in probablity distributions) are encoded in binary, except in Section 5. Moreover, we represent monotone neighbourhood systems $\mathfrak{A}$ by set systems $\mathfrak{A}_0 \subseteq \mathfrak{A}$ that generate $\mathfrak{A}$ by upwards closure; otherwise, representations are essentially obvious.

The *size* $\mathsf{size}(\xi)$ of a finite coalgebra $\xi : C \to T(C)$ is then defined as $\sum_{x \in C}(1+\mathsf{size}(\xi(x)))$. We also fix the input for the model checking algorithm: First, we fix a T-coalgebra (C, ξ) with C finite; second, we fix a closed target formula χ, assuming w.l.o.g. that χ is *clean*, i.e. that every fixpoint variable is bound by at most one fixpoint operator in χ. For a variable $X \in V$ that is bound in χ, we then write $\theta(X)$ to denote *the* formula $\eta X.\psi$ that is a subformula of χ. Let $\mathsf{Cl}(\chi)$ be the *closure* (that is, the set of subformulae) of χ. We have $|\mathsf{Cl}(\chi)| \leq |\chi| \leq |\mathsf{size}(\chi)|$, where $|\chi|$ denotes the number of operators or variables in χ (ignoring representation sizes). We put $k = \max\{\mathsf{ad}(\eta X.\psi) \mid \eta X.\psi \in \mathsf{Cl}(\chi)\}$.

3 Local Model Checking for the Coalgebraic μ-Calculus

We proceed to introduce and analyse our model checking algorithm which essentially is a more general version of the fixpoint iteration algorithm for standard parity games [3]. The algorithm will interface with the functor via the following computational problem:

▶ **Definition 2** (One-step satisfaction problem)**.** Let T be a functor and let C be a set. The *one-step satisfaction problem* for inputs $s \in T(C)$, $\heartsuit \in \Lambda$ and $U \subseteq C$ consists in deciding whether $s \in [\![\heartsuit]\!]U$. We denote the time it takes to solve the problem for input $s, \heartsuit, U$ by $t(\mathsf{size}(s), \mathsf{size}(\heartsuit), |C|)$, having $|U| \leq |C|$.

We discuss the one-step satisfaction problem for some of the logics from Example 1:

▶ **Example 3.**

1. In the relational case, with $T = \mathcal{P}$, $\Lambda = \{\Diamond, \Box\}$, to check one-step satisfaction for $s \in \mathcal{P}(C), \Diamond, U \subseteq C$, we have to check whether s intersects with U; and to check one-step satisfaction for $s, \Box, U$, we have to check whether s is a subset of U. So $t(\mathsf{size}(s), \mathsf{size}(\heartsuit), |C|) \leq |C|$ for $\heartsuit \in \Lambda$, assuming that containment in sets can be checked in constant time.
2. In the graded case, with $T = \mathcal{B}$, $\Lambda = \{\langle b \rangle, [b] \mid b \in \mathbb{N}\}$, to check one-step satisfaction for $s \in \mathcal{B}(C), \langle b \rangle, U \subseteq C$, we have to check whether $\sum_{u \in U} s(u) > b$; to check one-step satisfaction for $s, [b], U$, we have to check whether $\sum_{u \in (C \setminus U)} s(u) \leq b$. Hence $t(\mathsf{size}(s), \mathsf{size}(\heartsuit), |C|) \leq \mathsf{size}(\heartsuit) \cdot |C|$ for $\heartsuit \in \Lambda$; this holds even if grades are coded in binary since binary numbers can be added and compared in linear time.
3. In the probabilistic case, with $T = \mathcal{D}$, $\Lambda = \{\langle p \rangle, [p] \mid p \in \mathbb{Q} \cap [0, 1]\}$, to check one-step satisfaction for $s \in \mathcal{D}(C), \langle p \rangle, U \subseteq C$, we have to check whether $\sum_{u \in U} s(u) > p$; to check one-step satisfaction for $s, [p], U$, we have to check whether $\sum_{u \in (C \setminus U)} s(u) \leq p$. Hence $t(\mathsf{size}(s), \mathsf{size}(\heartsuit), |C|) \in \mathcal{O}(((|C| \cdot \mathsf{size}(s))^2) \cdot |C|)$ for $\heartsuit \in \Lambda$. This holds even if probabilities are coded in binary since quotients of o-bit binary numbers can be added in time $\mathcal{O}(o^2)$; there are at most $|C|$ summation steps, and since each summation step increases the number of bits of the denominator of the summand by at most $\mathsf{size}(s)$, each summation step can be done in time $(|C| \cdot \mathsf{size}(s))^2$.

Since our model checking algorithm can be seen as an algorithm for solving a coalgebraic variant of parity games, we now recall some basic notions of parity games (see e.g. [14]).

▶ **Definition 4** (Parity games). A *parity game* (V, E, α) consists of a set of *nodes* V, a set of *moves* $E \subseteq V \times V$ and a *priority function* $\alpha : V \to \mathbb{N}$. Furthermore, each node belongs to exactly one of the players Eloise or Abelard (where we denote Eloise's nodes by $V_\exists$ and Abelard's nodes by $V_\forall$). A *play* $\rho = v_0, v_1, \ldots \subseteq V^* \cup V^\omega$ is a (finite or infinite) sequence of nodes such that for all $i \geq 0$ such that ρ contains at least $i + 1$ nodes, we have $(v_i, v_{i+1}) \in E$. We say that an infinite play $\rho = v_0, v_1, \ldots$ is *even*, if the largest priority that occurs infinitely often in it is even (formally, if $\max\{\alpha(v) \mid \forall j. (v_j = v) \Rightarrow \exists j' > j. (v_{j'} = v) \text{ and } \exists j. v_j = v\}$ is an even number), and *odd* otherwise; finite plays are required to end in nodes that have no outgoing move. Player Eloise wins all even plays and finite plays that end in an Abelard-node; player Abelard wins all other plays. The *size* of a parity game (V, E, α) is $|V|$. A *(history-free)* Eloise-*strategy* $s : V_\exists \rightharpoonup V$ is a partial function that assigns moves $s(x)$ to Eloise-nodes $x \in \mathsf{dom}(s)$. A play $\rho = v_0, v_1, \ldots$ *follows* a strategy s if for all $i \geq 0$ such that $v_i \in V_\exists$, $v_{i+1} = s(v_i)$. An Eloise-strategy *wins* a node $v \in V$ if Eloise wins all plays that start at v and follow s. We have a dual notion of Abelard-strategies; *solving* a parity game consists in computing the *winning regions* of the two players, that is, the sets of states that they win.

A crucial property of parity games is that they are *history-free determined* [14], that is, that every node in a parity game is won by exactly one of the two players and then there is a history-free strategy for the respective player that wins the node.

One natural way to solve a parity game is by *fixpoint iteration* [3] (which essentially computes a particular *progress measure annotation* [18] of the game). Algorithms that use this method repeatedly apply a function (denoted by Ψ in Section 3.1 in [3] and sometimes also referred to as *is_mother*, see e.g. Section 4.2 in [20]) that evaluates the allowed moves at each node to compute the set of nodes that are won by player Eloise *under the assumption* that sets of nodes computed by previous iterations of the function are also won by her. Intuitively, the algorithm keeps a set of currently allowed nodes Y_i for each priority i in memory; one iteration of the function *is_mother* then computes the set of nodes *is_mother*$(Y_0, \ldots, Y_k)$

which have some priority j and which either belong to Eloise and have *some* move to a node from Y_j or belong to Abelard and *only* have moves to nodes from Y_j. The repeated application of this function is defined in terms of a nested fixpoint (referred to as Φ_d in [3], Section 3.1). The winning region for Abelard is computed as the dual fixpoint of the complementary function $\overline{is\_mother}$. Our model checking algorithm proceeds in a very similar fashion, but crucially uses instances of the one-step satisfaction problem to evaluate modal nodes. We proceed to define our variants of the is_mother and $\overline{is\_mother}$ functions which we call f and g for brevity in this work.

▶ **Definition 5.** For $V = \mathsf{Cl}(\chi) \times C$, we define the *tracing function* $h : V \to \mathcal{P}(V)$ by putting

$$h(\psi, x) = \begin{cases} \emptyset & \text{if } \psi = \bot \text{ or } \psi = \top \\ \{(\psi_1, x), (\psi_2, x)\} & \text{if } \psi = \psi_1 \vee \psi_2 \text{ or } \psi = \psi_1 \wedge \psi_2 \\ \{(\psi_1, x)\} & \text{if } \psi = \eta X.\psi_1 \\ \{(\theta(X), x)\} & \text{if } \psi = X \\ \{(\psi_1, y) \mid y \in C\} & \text{if } \psi = \heartsuit\psi_1 \end{cases}$$

A non-modal formula ψ can be traced to the formula ϕ at x if and only if $(\phi, x) \in h(\psi, x)$, so e.g. a conjunction $\psi_1 \wedge \psi_2$ can be traced to ψ_1 and to ψ_2, all at some state x. A modal formula $\heartsuit\psi_1$ can be traced from x to the formula ψ_1 at any state y (that is, we have $(\psi_1, y) \in h(\heartsuit\psi_1, x)$ for all $y \in C$); for instance, $\Diamond\psi$ can be traced from some state x to ψ at any state $y \in C$. Hence $h(\psi, x)$ computes the set of nodes that are relevant for the (one-step) satisfaction of ψ at x. For a given set $G \subseteq V$, we then define $k+1$-ary functions $f, g : (\mathcal{P}(G))^{k+1} \to \mathcal{P}(G)$ by putting, for $\mathbf{Y} = (Y_0, \dots, Y_k) \in (\mathcal{P}(G))^{k+1}$,

$$\begin{aligned} f(\mathbf{Y}) =& \{(\top, x) \mid (\top, x) \in G\} \cup \{(\heartsuit\psi, x) \in G \mid \xi(x) \in [\![\heartsuit]\!]\{y \mid (\psi, y) \in Y_0\}\} \cup \\ & \{(\psi \vee \phi, x) \in G \mid h(\psi \vee \phi, x) \cap Y_0 \neq \emptyset\} \cup \{(\psi \wedge \phi, x) \in G \mid h(\psi \wedge \phi, x) \subseteq Y_0\} \cup \\ & \{(\eta X.\, \psi, x) \in G \mid h(\eta X.\, \psi, x) \subseteq Y_0\} \cup \{(X, x) \mid h(X, x) \subseteq Y_{\mathsf{ad}(\theta(X))}\} \\ g(\mathbf{Y}) =& \{(\bot, x) \mid (\bot, x) \in G\} \cup \{(\heartsuit\psi, x) \in G \mid \xi(x) \notin [\![\heartsuit]\!]\{y \mid (\psi, y) \in (G \setminus Y_0)\}\} \cup \\ & \{(\psi \vee \phi, x) \in G \mid h(\psi \vee \phi, x) \subseteq Y_0\} \cup \{(\psi \wedge \phi, x) \in G \mid h(\psi \wedge \phi, x) \cap Y_0 \neq \emptyset\} \cup \\ & \{(\eta X.\, \psi, x) \in G \mid h(\eta X.\, \psi, x) \subseteq Y_0\} \cup \{(X, x) \in G \mid h(X, x) \subseteq Y_{\mathsf{ad}(\theta(X))}\}. \end{aligned}$$

Finally, we put

$$\mathbf{E}_G = \eta_k Y_k. \dots .\eta_1 Y_1.\eta_0 Y_0.f(\mathbf{Y}) \quad \text{and} \quad \mathbf{A}_G = \overline{\eta_k} Y_k. \dots .\overline{\eta_1} Y_1.\overline{\eta_0} Y_0.g(\mathbf{Y}),$$

where η_i is GFP if i is even and LFP otherwise and where $\overline{\mathsf{LFP}} = \mathsf{GFP}$ and $\overline{\mathsf{GFP}} = \mathsf{LFP}$.

For instance, we have $(X, x) \in f(\mathbf{Y})$ if $(\theta(X), x) \in Y_p$ where $p = \mathsf{ad}(\theta(X))$, that is, when passing the fixpoint variable X, priority p occurs in the game. We also have e.g. $(\psi \vee \phi, x) \in f(\mathbf{Y})$ if $(\psi, x) \in Y_0$ or $(\phi, x) \in Y_0$ but $(\psi \vee \phi, x) \in g(\mathbf{Y})$ if $(\psi, x) \in Y_0$ *and* $(\phi, x) \in Y_0$; the latter constraint comes with the intuition that g is used to derive *non*-satisfaction of formulas. Checking whether some pair $(\heartsuit\psi, x)$ is contained in $f(\mathbf{Y})$ or $g(\mathbf{Y})$ is an instance of the one-step satisfaction problem with input $\xi(x)$,$\heartsuit$,$\{y \mid (\psi, y) \in U\}$ for some $U \subseteq V$; since $\mathsf{size}(\xi(x)) \leq \mathsf{size}(C)$, $\mathsf{size}(\heartsuit) \leq \mathsf{size}(\chi)$ and $\{y \mid (\psi, y) \in U\} \subseteq C$, the instance can be solved in time $t(\mathsf{size}(C), \mathsf{size}(\chi), |C|)$.

The sets $\mathbf{E}_G$ and $\mathbf{A}_G$ can be seen as winning regions for the two players Eloise and Abelard in a coalgebraic parity game with set of nodes $G \subseteq V$ that is very similar to a model checking parity game but uses instances of the one-step satisfaction problem instead of modal moves.

Intuitively, Eloise has, for all nodes $(\psi, x) \in \mathbf{E}_G$ and for all disjunctions that are encountered when checking satisfaction of ψ at x, a choice such that the combination of her choices guarantees that no trace of ψ through ξ that starts at x unfolds some least fixpoint infinitely often without also unfolding a surrounding fixpoint infinitely often.

We now introduce our local model checking algorithm, which iteratively adds nodes from V to a growing set of nodes G and computes the sets $\mathbf{E}_G$ and $\mathbf{A}_G$ in optional intermediate model checking steps. The algorithm hence inherently supports *local model checking* [30] since the coalgebraic model checking game is constructed step by step and the partially constructed game can be solved on-the-fly, that is, at any time while building up the game, terminating as soon as one of the players has a strategy that wins the initial node in the game played over G. Since the model sizes in model checking tend to be exponential in the sizes of formulas, on-the-fly solving seems to be a valuable capability as it allows to cut down the search space of the algorithm; a concrete implementation of the algorithm and an evaluation of this aspect remains a task for future work though.

Algorithm 1: Local model checking.

To decide whether $x \in \llbracket \chi \rrbracket$, initialize $U = \{(\chi, x)\}$, $G = \emptyset$.

1. Expansion: pick *some* $(\psi, y) \in U$, add it to G and remove it from U, and add those pairs from $h(\psi, y)$ that are not already contained in G to U.
2. (Optional) Check: compute $\mathbf{E}_G$ and/or $\mathbf{A}_G$. If $(\chi, x) \in \mathbf{E}_G$, then return "yes", if $(\chi, x) \in \mathbf{A}_G$, then return "no".
3. Loop: if $U \neq \emptyset$, then continue with step 1).
4. Final Check: compute $\mathbf{E}_G$. If $(\chi, x) \in \mathbf{E}_G$, then return "yes", otherwise return "no".

In the final checking step, all nodes that are reachable from (χ, x) using h have been added to G so that $\mathbf{E}_G = G \setminus \mathbf{A}_G$, i.e. every node – including (χ, x) – is won by exactly one of the players; in the optional intermediate checking steps however, there may be nodes for which no player has a winning strategy yet.

The correctness statement then reads as follows.

▶ **Lemma 6.** *We have $(\chi, x) \in \mathbf{E}_V$ if and only if $x \in \llbracket \chi \rrbracket$.*

Proof sketch. Let $(\chi, x) \in \mathbf{E}_V$. We use lexically ordered vectors of natural numbers as a measure for the computation of the nested fixpoint $\mathbf{E}_V$ (similar in spirit to coalgebraic progress measures [16]) and then show $x \in \llbracket \chi \rrbracket$ by nested induction and coinduction, using the measure to ensure termination of the inductive parts of the proof. For the converse direction, let $x \in \llbracket \chi \rrbracket$. Again we use lexically ordered vectors of natural numbers as a measure, but this time for the satisfaction of fixpoint formulas in models. The proof of $x \in \mathbf{E}_V$ then is again by nested induction and coinduction, using the latter vectors as termination measure for the inductive parts. ◀

▶ **Lemma 7.** *The algorithm runs in time $\mathcal{O}(|V| \cdot t(\mathsf{size}(C), \mathsf{size}(\chi), |C|) \cdot |V|^{k+1})$.*

Proof. The runtime of the algorithm is dominated by the time it takes to compute $\mathbf{E}_V$. We have to compute a $k+1$-nested fixpoint which can be done by fixpoint iteration, that is, by computing $f(\mathbf{Y})$ for some $\mathbf{Y}$ at most $|V|^{k+1}$ times. A single computation of $f(\mathbf{Y})$ can be implemented to run in time $|V| \cdot t(\mathsf{size}(C), \mathsf{size}(\chi), |C|)$ since we have to solve the one-step satisfaction problem at most for all $(\heartsuit\psi, x) \in V$, that is, at most $|V|$ times. ◀

This yields the following bounds in concrete instances:

▶ **Corollary 8.** *We obtain the following upper time bounds for the model checking problems of the respective μ-calculi:*

1. *for the graded μ-calculus: $\mathcal{O}((|\chi| \cdot |C|^2 \cdot \mathsf{size}(\chi) \cdot (|\chi| \cdot |C|)^{k+1})$;*
2. *for the probabilistic μ-calculus: $\mathcal{O}((|\chi| \cdot |C|^4 \cdot (\mathsf{size}(C))^2 \cdot (|\chi| \cdot |C|)^{k+1})$;*

Proof. Immediate from Lemma 7 by the observations from Example 3. ◀

4 Coalgebraic Model Checking in NP ∩ coNP

As a side result, we now show that if the one-step satisfaction problem of a coalgebraic logic is in P, then the model checking problem of the μ-calculus over this logic is in NP ∩ coNP. We derive this result independently of our local model checking algorithm, with the help of the exponentially sized evaluation games for the coalgebraic μ-calculus from Definition 3.5. in [7], which we briefly recall below; the authors of [7] use a slightly different but equivalent formulation of the games that uses the Fischer-Ladner closure instead of $\mathsf{Cl}(\chi)$.

▶ **Definition 9** (Evaluation games, [7])**.** The *evaluation game* for χ is a parity game $\mathcal{E}_\chi = (W, E, \alpha)$ with set of nodes $W = V \cup (\{\heartsuit\psi \in \mathsf{Cl}(\chi)\} \times \mathcal{P}(C))$, set of moves $E \subseteq W \times W$ and priority function $\alpha : W \to \mathbb{N}$. For nodes $(\psi, x) \in V$ such that ψ is not a modal operator, we put $E(\psi, x) = h(\psi, x)$; for $\heartsuit\psi \in \mathsf{Cl}(\chi)$, we put

$$E(\heartsuit\psi, x) = \{(\heartsuit\psi, U) \mid U \subseteq C, \xi(x) \in [\![\heartsuit]\!](U)\},$$

and for $U \subseteq C$, we put $E(\heartsuit\psi, U) = \{(\psi, y) \mid y \in U\}$. Nodes $(\bot, x)$, $(\psi_1 \vee \psi_2, x)$ and $(\heartsuit\psi, x)$ belong to player Eloise and nodes $(\top, x)$, $(\psi_1 \wedge \psi_2, x)$ and $(\heartsuit\psi, U)$ belong to player Abelard; the ownership of nodes $(\eta X.\, \psi, x)$ and (X, x) is irrelevant since such nodes have exactly one outgoing move. All nodes $(\psi, x) \in V$ such that ψ is not a fixpoint variable and all nodes $(\psi, U) \in \mathsf{Cl}(\chi) \times \mathcal{P}(C)$ have priority 0; nodes $(X, x) \in V$ have priority $\alpha(X, x) = \mathsf{ad}(\theta(X))$.

For modal nodes $(\heartsuit\psi, x)$, Eloise has to pick a set U of states in such a way that $\xi(x)$ is contained in the predicate on $T(C)$ obtained by lifting U with $[\![\heartsuit]\!]$ (that is, $\xi(x) \in [\![\heartsuit]\!](U)$); player Abelard in turn can challenge, for all $y \in U$, whether ψ is satisfied at y, that is, he can move from $(\heartsuit\psi, U)$ to all nodes (y, ψ) with $y \in U$. As we have $V \leq |\chi| \cdot |C|$ and hence $W \leq |\chi|(|C| + 2^{|C|})$, the sizes of evaluation games are exponential in the number of states of the input models.

Under the stated assumptions on the one-step satisfaction problem, our algorithm can be understood as a method to solve these exponential-sized parity games in time p^k where p is a polynomial in the formula size and the model size, by avoiding a full unfolding of the game.

▶ **Lemma 10.** *We have $(\chi, x) \in \mathbf{E}_V$ if and only if* Eloise *wins the node (χ, x) in $\mathcal{E}_\chi$.*

Proof. Directly from Lemma 6 and Theorem 3.6 in [7]. ◀

We now obtain the announced criterion for model checking in NP ∩ coNP:

▶ **Theorem 11.** *If the one-step satisfaction problem for a coalgebraic logic is in* P*, then the model checking problem for the μ-calculus over this logic is in* NP ∩ coNP*.*

Proof sketch. We recall that the coalgebraic μ-calculus is closed under negation so that it suffices to show containment in NP. A history-free Eloise-strategy turns an exponential-sized evaluation game into a polynomial-sized graph since it picks, for each modal node, exactly one of the exponentially many moves that are available to Eloise. We nondeterministically guess whether the input formula is satisfied at the input state or not; depending on this guess, we then guess a history-free winning strategy in the evaluation game for the input formula or its negation, and verify that it indeed is a winning strategy for the respective game. This verification can be done in polynomial time since the guessed strategy turns the game into a graph of polynomial size and since the one-step satisfaction problem (which has to be solved once for each node in the graph) can be solved in polynomial time by assumption. ◀

5 Small model checking games

As we have recalled in Definition 9, the one-step satisfaction problem can also be encoded in terms of the evalution games from [7]; this particular encoding however involves guessing sets of states and hence leads to games of exponential size. We now introduce the notions of one-step satisfaction arenas and games, which we use as an alternative game-based means to decide one-step satisfaction of modal operators; for certain logics, including the monotone μ-calculus, the graded μ-calculus with grades coded in unary, and the alternating-time μ-calculus, this enables the construction of polynomial-size parity games for model checking, which can be fed directly to parity game solvers and profit automatically from advances in parity game solving.

▶ **Definition 12** (One-step satisfaction arenas)**.** Recall that T is a functor and Λ a set of modal operators. Let C be a set. A *one-step satisfaction arena* $\mathsf{A}_{\heartsuit,t}$ for $\heartsuit \in \Lambda$ and $t \in T(C)$ consists of a set $V_{\heartsuit,t}$ of nodes that is made up of

- the *initial node* $(\heartsuit, t)$,
- a set $I_{\heartsuit,t}$ of *inner nodes*,
- the set C of *exit nodes*,

and an *acyclic* set $E_{\heartsuit,t} \subseteq V_{\heartsuit,t} \times V_{\heartsuit,t}$ of moves such that $E_{\heartsuit,t}(x) = \emptyset$ for all exit nodes $x \in C$. Additionally, the initial and the inner nodes belong to exactly one of the players Eloise or Abelard.

▶ **Example 13.**

1. For $T = \mathcal{P}$, one-step satisfaction arenas have depth one and hence do not have inner nodes. The initial node $(\Diamond, t)$ belongs to Eloise and she can move to any state $y \in t$; formally, we put $E_{\Diamond,t}(\Diamond, t) = \{y \mid y \in t\}$.
2. For the (serial) monotone μ-calculus, one-step satisfaction arenas have depth two. As set of inner nodes, we choose $I_{\Diamond,t} = t$; the initial node $(\Diamond, t)$ belongs to player Eloise and all nodes from $I_{\Diamond,t}$ belong to player Abelard. The moves are defined by putting $E_{\Diamond,t}(\Diamond, t) = \{A \mid A \in t\}$ and $E_{\Diamond,t}(A) = \{y \mid y \in A\}$. The one-step satisfaction arenas thus have $|I_{\Diamond,t}| = |t| \in \mathcal{O}(\mathsf{size}(C)) \subseteq 2^{\mathcal{O}(|C|)}$ inner nodes.
3. For graded logic, the one-step satisfaction arena for $\langle b \rangle$ and $t \in \mathcal{B}(C)$ has the set $I_{\langle b \rangle,t} = \{1, \dots, |C| + 1\} \times \{0, \dots, b\} \times \{0, 1\}$ as inner nodes and there is a referee move from the initial node $(\langle b \rangle, t)$ to $(1, 0, 0) \in I_{\langle b \rangle,t}$. We assume a linear ordering on C and let v_n denote the n-th element in the according sequence of states. Nodes $(n, c, 0)$ belong to player Eloise and nodes $(n, c, 1)$ to player Abelard. We have moves

$$E_{\langle b \rangle,t}(n, c, 0) = \{(n, \min(b + 1, c + t(v_n)), 1), (n + 1, c, 0)\}$$
$$E_{\langle b \rangle,t}(n, c, 1) = \{v_n, (n + 1, c, 0)\},$$

where we assume $n \leq |C|$ in the first clause. Nodes $(|C| + 1, c, 0)$ have no successors and they belong to player Abelard if $c > b$ and to player Eloise if $c \leq b$. Note how we stop counting when the counter c reaches $b + 1$ by taking $\min(b + 1, c + t(v_n))$ as new counter.

The one-step satisfaction arena for $(\langle b\rangle, t)$ contains $|I_{\langle b\rangle,t}| = 2(|C|+1)(b+1) \in \mathcal{O}(|C| \cdot b)$ inner nodes. The estimate on $|I_{\langle b\rangle,t}|$ is thus linear in the size of $\langle b\rangle$ if grades are coded in unary, and exponential if grades are coded in binary.

4. For probabilistic logic, we proceed analogously to the graded case; however, the obtained one-step arenas are of exponential size, even when probabilities are coded in unary. We define the one-step satisfaction arena for $\langle p\rangle$ and t to have the set $I_{\langle p\rangle,t} = \{1, \dots, |C|+1\} \times P_t \times \{0,1\}$ as inner nodes, where $P_t = \{q \in \mathbb{Q} \mid q \leq p, \exists U \subseteq C.\ \sum_{u \in U} t(u) = q\} \cup \{*\}$ (that is, P_t is the set of probabilities $q \leq p$ that can be encountered when summing up any combination of probabilities that t assigns to states). There is a referee move from the initial node $(\langle p\rangle, t)$ to $(1,0,0) \in I_{\langle p\rangle,t}$. Again, we assume a linear ordering on C and let v_n denote the n-th element in the according sequence of states. Nodes $(n,c,0)$ belong to player Eloise and nodes $(n,c,1)$ to player Abelard. We have moves

$$E_{\langle p\rangle,t}(n,c,0) = \{(n, c \oplus t(v_n), 1), (n+1, c, 0)\} \qquad E_{\langle p\rangle,t}(n,c,1) = \{v_n, (n+1,c,0)\},$$

where we assume $n \leq |C|$ in the first clause. Here, $c \oplus t(v_n) = c + t(v_n)$ if $c \neq *$ and $c + t(v_n) \leq p$ and $c \oplus t(v_n) = *$ if $c = *$ or $c + t(v_n) > p$, that is, we stop counting once a probability greater than p has been reached. Nodes $(|C|+1, c, 0)$ have no successors and they belong to player Abelard if $c = *$ and to player Eloise if $c \neq *$. The one-step satisfaction arena for $(\langle p\rangle, t)$ contains $|I_{\langle p\rangle,t}| = 2(|C|+1)|P_t| \in \mathcal{O}(|C| \cdot |P_t|)$ inner nodes; we have $|P_t| \leq 2^{|C|} + 1$.

5. For alternating-time logic, let $N = \{1, \dots, n\}$ be the set of agents. For the one-step satisfaction arena for $([D], t)$ with $D \subseteq N$, $t = (k_1, \dots, k_n, f) \in \mathcal{G}(C)$, and $f : (\prod_{i \in N}[k_i]) \to C$, we put $I_{[D],t} = S_D$. The arena has the initial node $([D], t)$ that belongs to player Eloise while all inner nodes $s_D \in S_D$ belong to player Abelard. The moves are defined by

$$E_{[D],t}([D], t) = \{s_D \mid s_D \in S_D\} \qquad E_{[D],t}(s_D) = \{f(s_D, s_{N \setminus D}) \mid s_{N \setminus D} \in S_{N \setminus D}\}.$$

The one-step satisfaction arena for $([D], t)$ has $|I_{[D],t}| = |S_D| \in \mathcal{O}(\mathsf{size}(C))$ inner nodes.

6. Also, we can always take the general one-step arena with initial node $(\heartsuit, t)$, belonging to Eloise, set of inner nodes $I_{\heartsuit,t} = \{U \subseteq C \mid t \in [\![\heartsuit]\!](U)\}$ (having $|I_{\heartsuit,t}| \leq 2^{|C|}$), all belonging to Abelard, and with moves $E_{\heartsuit,t}(\heartsuit, t) = \{U \subseteq C \mid t \in [\![\heartsuit]\!](U)\}$ and $E_{\heartsuit,t}(U) = \{y \mid y \in U\}$.

In all examples, except the last one, the one-step satisfaction arenas for dual modal operators are obtained by simply switching the ownership of nodes.

▶ **Definition 14** (One-step games). A *one-step game* $(\mathsf{A}_{\heartsuit,t}, U)$ consists of a one-step satisfaction arena $\mathsf{A}_{\heartsuit,t}$ for $\heartsuit$ and t together with a set $U \subseteq C$ of states, encoding a winning condition; player Eloise *wins* the game $(\mathsf{A}_{\heartsuit,t}, U)$ if she has a strategy s such that all plays in $\mathsf{A}_{\heartsuit,t}$ that start at $(\heartsuit, t)$ and that are played according to s end in exit nodes $c \in U$ or in inner nodes that belong to Abelard. A one-step satisfaction arena $\mathsf{A}_{\heartsuit,t}$ for $\heartsuit$ and t is *one-step sound and complete* if for all $U \subseteq C$, we have $t \in [\![\heartsuit]\!]U$ if and only if Eloise wins the one-step game $(\mathsf{A}_{\heartsuit,t}, U)$.

▶ **Example 15.** The one-step satisfaction arenas from Example 13 all are one-step sound and complete:

1. In the relational case, let $t \in \mathcal{P}(C)$ and $U \subseteq C$. We have $t \in [\![\Diamond]\!]U$ if and only if $t \cap U \neq \emptyset$ which in turn is the case if and only if there is some $y \in t$ such that $y \in U$. Since Eloise can move from $(\Diamond, t)$ to y if and only if $y \in t$, the last statement is true if and only if Eloise wins the game $(\mathsf{A}_{\Diamond,t}, U)$.

2. For the (serial) mononotone μ-calculus, let $t \in \mathcal{P}(\mathcal{P}(C))$ and $U \subseteq C$. We have $t \in [\![\Diamond]\!]U$ if and only if there is some $A \in t$ such that $A \subseteq U$. The latter is the case if and only if there is some $A \in t$ (so that Eloise can move from $(\Diamond, t)$ to A) such that for all $y \in A$, we have $y \in U$ and hence an according Abelard-move from B to the exit node y. The move from $(\Diamond, t)$ to A thus constitutes a winning strategy for Eloise.
3. For graded logic, assume input $\langle b \rangle$ where $b \in \mathbb{N}$ and $t \in \mathcal{B}(C)$ and let $U \subseteq C$. We have $t \in [\![\langle b \rangle]\!]U$ if and only if $\sum_{a \in U} t(a) > b$. So let $\sum_{a \in U} t(a) > b$. The function s defined by $s(n, c, 0) = (n, \min(b+1, c+t(v_n)), 1)$ if $v_n \in U$ and $s(n, c, 0) = (n+1, c, 0)$ if $v_n \notin U$ is a strategy that ensures that all plays of the game end at exit nodes $y \in U$ or at the inner node $s(|C|+1, \sum_{a \in U} t(a), 0)$ which belongs to Abelard. The former is the case since s uses exactly nodes $v_n \in U$ to increase the counter so that Abelard can only reach exit nodes from U. The latter is the case since there is just one play that reaches an ending node $s(|C|+1, c, 0)$ for some c and during this play, s moves in such a way that the counter sums up the $t(a)$ for all $a \in U$. For the converse direction, let Eloise have a winning strategy in the game $(\mathsf{A}_{\langle b \rangle, t}, U)$. Using this strategy, Eloise only uses states $v_n \in U$ to increase the counter; there is just a single final inner node of the shape $(|C|+1, c, 0)$ to which the strategy can lead and this node belongs to Abelard by assumption so that we have $c > b$. There is some set $A \subseteq U$ such that if the play that leads to this final node, the counter sums up all $t(a)$ for $a \in A$. Thus we have we have $c = t(A) \leq t(U)$ and hence $t(U) > b$, as required.
4. For probabilistic logic, the proof is analogous to the proof for graded logic.
5. For alternating-time logic, assume input $[D]$ where $D \subseteq \{1, \dots, n\}$ and $t \in \mathcal{G}(C)$ where $t = (k_1, \dots, k_n, f)$ and let $U \subseteq C$. We have $t \in [\![[D]]\!]U$ if and only if there is some $s_D \in S_D$ such that for all $s_{N \setminus D} \in S_{N \setminus D}$, we have $f(s_D, s_{N \setminus D}) \in U$. So assume that the latter is the case. The move from $([D], t)$ to s_D constitutes a winning strategy for Eloise since for all Abelard-moves from s_D to some exit node $f(s_D, s_{N \setminus D})$, we have $f(s_D, s_{N \setminus D}) \in U$ by assumption. For the converse direction, let there be a winning strategy for Eloise in the game $(\mathsf{A}_{[D], t}, U)$ that moves from $([D], t)$ to some s_D such that for all Abelard-moves from s_D to some $f(s_D, s_{N \setminus D})$, we have $f(s_D, s_{N \setminus D}) \in U$. Then we have $\exists s_D \in S_D . \forall s_{N \setminus D} \in S_{N \setminus D} . f(s_D, s_{N \setminus D}) \in U$.
6. The general one-step arenas from Example 13.6 are easily seen to be one-step sound and complete. However, when using these one-step arenas, the model checking games from Definition 16 below are essentially just the evaluation games recalled in Definition 9.

In the following, we assume that one-step sound and complete one-step satisfaction arenas are available for all $\heartsuit$ and t and that the sets of inner nodes of all these arenas are disjoint. We continue to define our modular model checking game by plugging the one-step satisfaction arenas into the modal steps of a standard parity model checking game:

▶ **Definition 16** (Coalgebraic model checking games)**.** The *model checking game* $\mathcal{G}_\chi = (W, E, \alpha)$ for χ is a parity game that is played over the set of nodes

$$V' = V \cup \bigcup_{\heartsuit\psi \in \mathsf{Cl}(\chi), x \in C} (\{\psi\} \times V_{\heartsuit, \xi(x)}).$$

The set of moves $E \subseteq W \times W$ is defined by putting $E(\psi, x) = h(\psi, x)$ if $(\psi, x) \in V$ and ψ is not a modal operator; for $(\heartsuit\psi, x) \in V$, we put $E(\heartsuit\psi, x) = \{(\psi, (\heartsuit, \xi(x)))\}$, that is, there is a referee move from such nodes to the initial node of the one-step arena for $\heartsuit$ and $\xi(x)$, paired with the formula ψ. For nodes $(\psi, v) \in \{\psi\} \times V_{\heartsuit, \xi(x)}$ such that $v \in V_{\heartsuit, \xi(x)}$ is not an exit node, we put $E(\psi, v) = \{\psi\} \times E_{\heartsuit, \xi(x)}(v)$, that is, at such nodes, the moves are inherited from the arena $\mathsf{A}_{\heartsuit,\xi(x)}$ and take place only in the second component. Nodes $(\top, x)$ and $(\psi_1 \wedge \psi_2, x)$ belong to player Abelard, nodes $(\bot, x)$ and $(\psi_1 \vee \psi_2, x)$ belong to player Eloise. The ownership of nodes $(\eta X.\psi, x)$, (X, x) and $(\heartsuit\psi, x)$ is irrelevant. Nodes $(\psi, v) \in \{\psi\} \times V_{\heartsuit,\xi(x)}$ such that $v \in V_{\heartsuit,\xi(x)}$ is not an exit node are owned by the player that owns v in the arena $\mathsf{A}_{\heartsuit,\xi(x)}$. All nodes, including the nodes from the one-step satisfaction arenas, have priority 0, with the exception of nodes $(X, x) \in V$ which have priority $\alpha(X, x) = \mathsf{ad}(\theta(X))$.

The game $\mathcal{G}_\chi$ is a parity game with k priorities and $|V| + \sum_{\heartsuit\psi\in\mathsf{Cl}(\chi),x\in C} |I_{\heartsuit,\xi(x)}|$ nodes.

▶ **Theorem 17.** *We have $x \in [\![\chi]\!]$ if and only if* Eloise *wins the node (χ, x) in $\mathcal{G}_\chi$.*

Proof. We have that Eloise wins the node (χ, x) in $\mathcal{G}_\chi$ if and only if Eloise wins the node (χ, x) in the evaluation game (recalling Definition 9): the only difference between $\mathcal{G}_\chi$ and the evaluation game for χ is the modal step. If Eloise wins a node $(\heartsuit\psi, x)$ in the evaluation game, then there is a set U such that $\xi(x) \in [\![\heartsuit]\!](U)$ and Eloise wins all nodes (ψ, y) such that $y \in U$. It suffices to show that Eloise can win the one-step satisfaction game $(\mathsf{A}_{\heartsuit,\xi(x)}, U)$; this however is the case since the one-step arena $\mathsf{A}_{\heartsuit,\xi(x)}$ for $\heartsuit$ and $\xi(x)$ is one-step complete. Conversely, let Eloise win a node $(\heartsuit\psi, x)$ in $\mathcal{G}_\chi$. Then there is some set $U \subseteq C$ of exit nodes such that Eloise wins all nodes from $\{\psi\} \times U$ as well as the one-step game $(\mathsf{A}_{\heartsuit,\xi(x)}, U)$. By one-step soundness of the one-step arena $\mathsf{A}_{\heartsuit,\xi(x)}$, we have $\xi(x) \in [\![\heartsuit]\!](U)$ so that in the evaluation game, Eloise can move from $(\heartsuit\psi, x)$ to $(\heartsuit\psi, U)$ and for each Abelard-move from $(\heartsuit\psi, U)$ to (ψ, y) such that $y \in U$, Eloise wins (ψ, y). Lemmas 10 and 6 – or alternatively, Theorem 3.6 in [7] – finish the proof. ◀

Using parity games, we can decide the model checking problem in quasi-polynomial time if the one-step satisfaction arenas are of at most quasi-polynomial size.

▶ **Corollary 18.** *Let $q(\mathsf{size}(\chi), \mathsf{size}(C))$ denote the maximal number of inner nodes of the one-step satisfaction arenas that are used when constructing the model checking game for χ and C. Model checking C against χ can be done in time $\mathcal{O}(((|\chi| \cdot |C|) \cdot q(\mathsf{size}(\chi), \mathsf{size}(C)))^{\log(k)+6})$.*

Proof. The model checking game $\mathcal{G}_\chi$ consists of at most $|\chi| \cdot |C|$ one-step satisfaction games that are plugged into one parity game of size $|\chi| \cdot |C|$, resulting in a parity game of size $\mathcal{O}((|\chi| \cdot |C|) \cdot q(\mathsf{size}(\chi), \mathsf{size}(C)))$ and with k priorities. Using the methods from [4], this game can be solved in time $\mathcal{O}(((|\chi| \cdot |C|) \cdot q(\mathsf{size}(\chi), \mathsf{size}(C)))^{\log(k)+6})$. ◀

▶ **Corollary 19.** *We obtain the following quasi-polynomial upper time bounds for the model checking problems of the respective μ-calculi:*

1. *for the relational μ-calculus: $\mathcal{O}((|\chi| \cdot |C|)^{\log(k)+6})$;*
2. *for the (serial) monotone μ-calculus: $\mathcal{O}((|\chi| \cdot |C| \cdot \mathsf{size}(C))^{\log(k)+6})$;*
3. *for the graded μ-calculus with grades coded in unary: $\mathcal{O}((|\chi| \cdot |C|^2 \cdot \mathsf{size}(\chi))^{\log(k)+6})$;*
4. *for the alternating-time μ-calculus: $\mathcal{O}((|\chi| \cdot |C| \cdot \mathsf{size}(C))^{\log(k)+6})$.*

Proof. Immediate from Corollary 18 by the observations from Example 13. ◀

6 Conclusion

We have presented an algorithm to solve the model checking problem for coalgebraic μ-calculi in time p^k, where p is a polynomial in the input formula size and the input model size and where k is the alternation depth of the input formula; while the algorithm is correct for all instances of the coalgebraic μ-calculus, the runtime analysis relies on the assumption that the one-step satisfaction problem of the base logic is in P. This holds in many instance logics, including the graded μ-calculus and the probabilistic μ-calculus, even if grades and probabilities are coded in binary. We also have shown that under the same assumption on the one-step satisfaction problem, the model checking problem of a coalgebraic μ-calculus is in NP $\cap$ coNP. Our approach relies centrally on a coalgebraic variant of parity games, whose algorithmics we will develop further in future research. For certain logics, including the graded μ-calculus with grades coded in unary, the alternating-time μ-calculus and the monotone μ-calculus, we have shown how to construct model checking games as *standard* parity games of polynomial size; this enables the efficient use of standard parity game solvers in model checking, and also transfers any future progress on solving parity games directly to model checking for such logics.

References

1 Rajeev Alur, Thomas Henzinger, and Orna Kupferman. Alternating-time temporal logic. *J. ACM*, 49:672–713, 2002.

2 Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* MIT Press, 2008.

3 Florian Bruse, Michael Falk, and Martin Lange. The Fixpoint-Iteration Algorithm for Parity Games. In *Games, Automata, Logics and Formal Verification, GandALF 2014*, volume 161 of *EPTCS*, pages 116–130, 2014.

4 Cristian Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. Deciding parity games in quasipolynomial time. In *Theory of Computing, STOC 2017*, pages 252–263. ACM, 2017.

5 Brian F. Chellas. *Modal Logic.* Cambridge University Press, 1980.

6 Corina Cîrstea, Clemens Kupke, and Dirk Pattinson. EXPTIME tableaux for the coalgebraic μ-calculus. In *Computer Science Logic, CSL 2009*, volume 5771 of *LNCS*, pages 179–193. Springer, 2009.

7 Corina Cîrstea, Clemens Kupke, and Dirk Pattinson. EXPTIME tableaux for the coalgebraic μ-calculus. *Log. Meth. Comput. Sci.*, 7, 2011.

8 Corina Cîrstea, Alexander Kurz, Dirk Pattinson, Lutz Schröder, and Yde Venema. Modal logics are coalgebraic. *Comput. J.*, 54:31–41, 2011.

9 Corina Cîrstea, Shunsuke Shimizu, and Ichiro Hasuo. Parity Automata for Quantitative Linear Time Logics. In *Algebra and Coalgebra in Computer Science, CALCO 2017*, volume 72 of *LIPIcs*, pages 7:1–7:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. URL: `http://www.dagstuhl.de/dagpub/978-3-95977-033-0`.

10 Giovanna D'Agostino and Albert Visser. Finality regained: A coalgebraic study of Scott-sets and multisets. *Arch. Math. Logic*, 41:267–298, 2002.

11 E. Allen Emerson, Charanjit Jutla, and A. Prasad Sistla. On model checking for the μ-calculus and its fragments. *Theor. Comput. Sci.*, 258:491–522, 2001. URL: `http://dx.nodoi.org/10.1016/S0304-3975(00)00034-7`.

12 Sebastian Enqvist, Fatemeh Seifan, and Yde Venema. Monadic Second-Order Logic and Bisimulation Invariance for Coalgebras. In *Logic in Computer Science, LICS 2015*. IEEE, 2015.

13 Alessandro Ferrante, Aniello Murano, and Mimmo Parente. Enriched μ-Calculi Module Checking. *Log. Methods Comput. Sci.*, 4(3), 2008.

14 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`.

15 Helle Hvid Hansen, Clemens Kupke, Johannes Marti, and Yde Venema. Parity Games and Automata for Game Logic. In *Dynamic Logic. New Trends and Applications, DALI 2017*, volume 10669 of *LNCS*, pages 115–132. Springer, 2018.

16 Ichiro Hasuo, Shunsuke Shimizu, and Corina Cîrstea. Lattice-theoretic Progress Measures and Coalgebraic Model Checking. In *Principles of Programming Languages, POPL 2016*, pages 718–732. ACM, 2016.

17 Daniel Hausmann and Lutz Schröder. Optimal Satisfiability Checking for Arithmetic μ-Calculi. In *Foundations of Software Science and Computation Structures, FOSSACS 2019*, volume 11425 of *LNCS*, pages 277–294. Springer, 2019.

18 Marcin Jurdziński. Small Progress Measures for Solving Parity Games. In Horst Reichel and Sophie Tison, editors, *Symposium on Theoretical Aspects of Computer Science, STACS 2000*, pages 290–301, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

19 Dexter Kozen. Results on the propositional μ-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

20 Orna Kupferman, Ulrike Sattler, and Moshe Vardi. The Complexity of the Graded μ-Calculus. In *Automated Deduction, CADE 02*, volume 2392 of *LNCS*, pages 423–437. Springer, 2002.

21 Wanwei Liu, Lei Song, Ji Wang, and Lijun Zhang. A Simple Probabilistic Extension of Modal Mu-calculus. In *International Joint Conference on Artificial Intelligence, IJCAI 2015*, pages 882–888. AAAI Press, 2015.

22 Damian Niwinski. On Fixed-Point Clones (Extended Abstract). In *Automata, Languages and Programming, ICALP 1986*, volume 226 of *LNCS*, pages 464–473. Springer, 1986.

23 Rohit Parikh. The logic of games and its applications. *Ann. Discr. Math.*, 24:111–140, 1985.

24 Marc Pauly. A Modal Logic for Coalitional Power in Games. *J. Logic Comput.*, 12:149–166, 2002.

25 David Peleg. Concurrent dynamic logic. *J. ACM*, 34:450–479, 1987. URL: `http://nodoi.acm.org/10.1145/23005.23008`.

26 Jan Rutten. Universal Coalgebra: A Theory of Systems. *Theor. Comput. Sci.*, 249:3–80, 2000.

27 Philippe Schnoebelen. The Complexity of Temporal Logic Model Checking. In *Advances in Modal Logic, AiML 2002*, pages 393–436. College Publications, 2003.

28 Lutz Schröder and Dirk Pattinson. Strong completeness of coalgebraic modal logics. In *Theoretical Aspects of Computer Science, STACS 09*, pages 673–684. Schloss Dagstuhl – Leibniz-Zentrum für Informatik; Dagstuhl, Germany, 2009.

29 Lutz Schröder and Yde Venema. Completeness of Flat Coalgebraic Fixpoint Logics. *ACM Trans. Comput. Log.*, 19(1):4:1–4:34, 2018.

30 Colin Stirling and David Walker. Local model checking in the modal mu-calculus. *Theoretical Computer Science*, 89(1):161–177, 1991.

Graded Monads and Graded Logics for the Linear Time – Branching Time Spectrum

Ulrich Dorsch
Friedrich-Alexander-Universität Erlangen-Nürnberg, Germany

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

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

Abstract

State-based models of concurrent systems are traditionally considered under a variety of notions of process equivalence. In the case of labelled transition systems, these equivalences range from trace equivalence to (strong) bisimilarity, and are organized in what is known as the linear time – branching time spectrum. A combination of universal coalgebra and graded monads provides a generic framework in which the semantics of concurrency can be parametrized both over the branching type of the underlying transition systems and over the granularity of process equivalence. We show in the present paper that this framework of *graded semantics* does subsume the most important equivalences from the linear time – branching time spectrum. An important feature of graded semantics is that it allows for the principled extraction of characteristic modal logics. We have established invariance of these *graded logics* under the given graded semantics in earlier work; in the present paper, we extend the logical framework with an explicit propositional layer and provide a generic expressiveness criterion that generalizes the classical Hennessy-Milner theorem to coarser notions of process equivalence. We extract graded logics for a range of graded semantics on labelled transition systems and probabilistic systems, and give exemplary proofs of their expressiveness based on our generic criterion.

2012 ACM Subject Classification Theory of computation → Concurrency; Theory of computation → Modal and temporal logics

Keywords and phrases Linear Time, Branching Time, Monads, System Equivalences, Modal Logics, Expressiveness

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.36

Related Version Full version with all proof details available at `https://arxiv.org/abs/1812.01317`.

Funding This work forms part of the DFG-funded project COAX (MI 717/5-2 and SCHR 1118/12-2).

1 Introduction

State-based models of concurrent systems are standardly considered under a wide range of system equivalences, typically located between two extremes respectively representing *linear time* and *branching time* views of system evolution. Over labelled transition systems, one specifically has the well-known *linear time – branching time spectrum* of system equivalences between trace equivalence and bisimilarity [42]. Similarly, e.g. probabilistic automata have been equipped with various semantics including strong bisimilarity [29], probabilistic (convex) bisimilarity [38], and distribution bisimilarity (e.g. [11, 16]). New equivalences keep appearing in the literature, e.g. for non-deterministic probabilistic systems [5, 43].

This motivates the search for unifying principles that allow for a generic treatment of process equivalences of varying degrees of granularity and for systems of different branching types (non-deterministic, probabilistic etc.). As regards the variation of the branching type,

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 36; pp. 36:1–36:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

universal coalgebra [35] has emerged as a widely-used uniform framework for state-based systems covering a broad range of branching types including besides non-deterministic and probabilistic, or more generally weighted, branching also, e.g., alternating, neighbourhood-based, or game-based systems. It is based on modelling the system type as an endofunctor on some base category, often the category of sets.

Unified treatments of system equivalences, on the other hand, are so far less well-established, and their applicability is often more restricted. Existing approaches include coalgebraic trace semantics in Kleisli [18] and Eilenberg-Moore categories [5,6,23,26,39,43], respectively. Both semantics are based on decomposing the coalgebraic type functor into a monad, the *branching type*, and a functor, the *transition type* (in different orders), and require suitable distributive laws between these parts; correspondingly, they grow naturally out of the functor but on the other hand apply only to functors that admit the respective form of decomposition. In the present work, we build on a more general approach introduced by Pattinson and two of us, based on mapping the coalgebraic type functor into a *graded monad* [31]. Graded monads correspond to algebraic theories where operations come with an explicit notion of *depth*, and allow for a stepwise evaluation of process semantics. Maybe most notably, graded monads systematically support a reasonable notion of *graded logic* where modalities are interpreted as *graded algebras* for the given graded monad. This approach applies to all cases covered in the mentioned previous frameworks, and additional cases that do not fit any of the earlier setups. We emphasize that graded monads are geared towards *inductively* defined equivalences such as finite trace semantics and finite-depth bisimilarity; we leave a similarly general treatment of infinite-depth equivalences such as infinite trace equivalence and unbounded-depth bisimilarity to future work. To avoid excessive verbosity, we restrict to models with finite branching throughout. Under finite branching, finite-depth equivalences typically coincide with their infinite-depth counterparts, e.g. states of finitely branching labelled transition systems are bisimilar iff they are finite-depth bisimilar, and infinite-trace equivalent iff they are finite-trace equivalent.

Our goal in the present work is to illustrate the level of generality achievable by means of graded monads in the dimension of system equivalences. We thus pick a fixed coalgebraic type, that of labelled transition systems, and elaborate how a number of equivalences from the linear time – branching time spectrum are cast as graded monads. In the process, we demonstrate how to extract logical characterizations of the respective equivalences from most of the given graded monads. For the time being, none of the logics we find are sensationally new, and in fact van Glabbeek already provides logical characterizations in his exposition of the linear time – branching time spectrum [42]; an overview of characteristic logics for non-deterministic and probabilistic equivalences is given by Bernardo and Botta [2]. The emphasis in the examples is mainly on showing how the respective logics are developed uniformly from general principles.

Using these examples as a backdrop, we develop the theory of graded monads and graded logics further. In particular,

- we give a more economical characterization of depth-1 graded monads involving only two functors (rather than an infinite sequence of functors);
- we extend the logical framework by a treatment of propositional operators – previously regarded as integrated into the modalities – as first class citizens;
- we prove, as our main technical result, a generic expressiveness criterion for graded logics guaranteeing that inequivalent states are separated by a trace formula.

Our expressiveness criterion subsumes, at the branching-time end of the spectrum, the classical Hennessy-Milner theorem [19] and its coalgebraic generalization [33,36] as well as expressiveness of probabilistic modal logic with only conjunction [12]; we show that it also covers expressiveness of the respective graded logics for more coarse-grained equivalences along the linear time – branching time spectrum. To illustrate generality also in the branching type, we moreover provide an example in a probabilistic setting, specifically we apply our expressiveness criterion to show expressiveness of a quantitative modal logic for probabilistic trace equivalence.

Related Work. Fahrenberg and Legay [17] characterize equivalences on the linear time – branching time spectrum by suitable classes of modal transition systems. We have already mentioned previous work on coalgebraic trace semantics in Kleisli and Eilenberg-Moore categories [5,6,18,23,26,39,43]. A common feature of these approaches is that, more precisely speaking, they model *language* semantics rather than trace semantics – i.e. they work in settings with explicit successful termination, and consider only successfully terminating traces. When we say that graded monads apply to all scenarios covered by these approaches, we mean more specifically that the respective language semantics are obtained by a further canonical quotienting of our trace semantics [31]. Having said that graded monads are strictly more general than Kleisli and Eilenberg-Moore style trace semantics, we hasten to add that the more specific setups have their own specific benefits including final coalgebra characterizations and, in the Eilenberg-Moore setting, generic determinization procedures. A further important piece of related work is Klin and Rot's method of defining trace semantics via the choice of a particular flavour of trace logic [28]. In a sense, this approach is opposite to ours: A trace logic is posited, and then two states are declared equivalent if they satisfy the same trace formulae. In our approach via graded monads, we instead pursue the ambition of first fixing a semantic notion of equivalence, and then designing a logic that characterizes this equivalence. Like Klin and Rot, we view trace equivalence as an inductive notion, and in particular limit attention to finite traces; coalgebraic approaches to infinite traces exist, and mostly work within the Kleisli-style setup [7–10,20,25,41]. Jacobs, Levy and Rot [22] use corecursive algebras to provide a unifying categorical view on the above-mentioned approaches to traces via Kleisli- and Eilenberg-Moore categories and trace logics, respectively. This framework does not appear to subsume the approach via graded monads, and like for the previous approaches we are not aware that it covers semantics from the linear time – branching time spectrum other than the end points (bisimilarity and trace equivalence).

2 Preliminaries: Coalgebra

We recall basic definitions and results in *(universal) coalgebra* [35], a framework for the unified treatment of a wide range of reactive systems. We write $1 = \{\star\}$ for a fixed one-element set, and $!\colon X \to 1$ for the unique map from a set X into 1. We write $f \cdot g$ for the composite of maps $g\colon X \to Y$, $f\colon Y \to Z$, and $\langle f, g\rangle\colon X \to Y \times Z$ for the pair map $x \mapsto (f(x), g(x))$ formed from maps $f\colon X \to Y$, $g\colon X \to Z$.

Coalgebra encapsulates the branching type of a given species of systems as a *functor*, for purposes of the present paper on the category of sets. Such a functor $G\colon \mathbf{Set} \to \mathbf{Set}$ assigns to each set X a set GX, whose elements we think of as structured collections over X, and to each map $f\colon X \to Y$ a map $Gf\colon GX \to GY$, preserving identities and composition. E.g. the *(covariant) powerset functor* $\mathcal{P}$ assigns to each set X the powerset $\mathcal{P}X$ of X, and to each map $f\colon X \to Y$ the map $\mathcal{P}f\colon \mathcal{P}X \to \mathcal{P}Y$ that takes direct images. (We mostly omit the description of the action of functors on maps in the sequel.) Systems with branching type described by G are then abstracted as *G-coalgebras*, i.e. pairs (X, γ) consisting of a set X of *states* and a map $\gamma\colon X \to GX$, the *transition map*, which assigns to each state $x \in X$ a structured collection $\gamma(x)$ of successors. For instance, a $\mathcal{P}$-coalgebra assigns to each state a set of successors, and thus is the same as a transition system.

▶ **Example 2.1.**

1. Fix a set $\mathcal{A}$ of *actions*. The functor $\mathcal{A} \times (-)$ assigns to each set X the set $\mathcal{A} \times X$; composing this functor with the powerset functor, we obtain the functor $G = \mathcal{P}(\mathcal{A} \times (-))$ whose coalgebras are precisely labelled transition systems (LTS): A G-coalgebra assigns to each state x a set of pairs (σ, y), indicating that y is a successor of x under the action σ.
2. The *(finite) distribution functor* $\mathcal{D}$ maps a set X to the set of finitely supported discrete probability distributions on X. These can be represented as probability mass functions $\mu\colon X \to [0,1]$, with $\sum_{x \in X} \mu(x) = 1$ and with the *support* $\{x \in X \mid \mu(x) > 0\}$ being finite. Coalgebras for $\mathcal{D}$ are precisely Markov chains. Composing with $\mathcal{A} \times (-)$ as above, we obtain the functor $\mathcal{D}(\mathcal{A} \times (-))$, whose coalgebras are *generative probabilistic transition systems*, i.e. assign to each state a distribution over pairs consisting of an action and a successor state.

As indicated in the introduction, we restrict attention to *finitary* functors G, in which every element $t \in GX$ is represented using only finitely many elements of X; formally, each set GX is the union of all sets $Gi_Y[GY]$ where Y ranges over finite subsets of X and i_Y denotes the injection $i_Y\colon Y \hookrightarrow X$. Concretely, this means that we restrict the set $\mathcal{A}$ of actions to be finite, and work with the *finite powerset functor* $\mathcal{P}_\omega$ (which maps a set X to the set of its finite subsets) in lieu of $\mathcal{P}$. ($\mathcal{D}$ as defined above is already finitary.)

Coalgebra comes with a natural notion of *behavioural equivalence* of states. A *morphism* $f\colon (X, \gamma) \to (Y, \delta)$ of G-coalgebras is a map $f\colon X \to Y$ that commutes with the transition maps, i.e. $\delta \cdot f = Gf \cdot \gamma$. Such a morphism is seen as preserving the behaviour of states (that is, behaviour is defined as being whatever is preserved under morphisms), and consequently states $x \in X$, $z \in Z$ in coalgebras (X, γ), (Z, ζ) are *behaviourally equivalent* if there exist coalgebra morphisms $f\colon (X, \gamma) \to (Y, \delta)$, $g\colon (Z, \zeta) \to (Y, \delta)$ such that $f(x) = g(z)$. For instance, states in LTSs are behaviourally equivalent iff they are bisimilar in the standard sense, and similarly, behavioural equivalence on generative probabilistic transition systems coincides with the standard notion of probabilistic bisimilarity [27]. We have an alternative notion of finite-depth behavioural equivalence: Given a G-coalgebra (X, γ), we define a series of maps $\gamma_n\colon X \to G^n 1$ inductively by taking γ_0 to be the unique map $X \to 1$, and $\gamma_{n+1} = G\gamma_n \cdot \gamma$. (These are the first ω steps of the *canonical cone* from X into the *final sequence* of G [1].) Then states x, y in coalgebras (X, γ), (Z, ζ) are *finite-depth behaviourally equivalent* if $\gamma_n(x) = \zeta_n(y)$ for all n; in the case where G is finitary, finite-depth behavioural equivalence coincides with behavioural equivalence [44].

3 Graded Monads and Graded Theories

We proceed to recall background on system semantics via graded monads introduced in our previous work [31]. We formulate some of our results over general base categories $\mathbf{C}$, using basic notions from category theory [30, 34]; for the understanding of the examples, it will suffice to think of $\mathbf{C} = \mathbf{Set}$. Graded monads were originally introduced by Smirnov [40] (with grades in a commutative monoid, which we instantiate to the natural numbers):

▶ **Definition 3.1** (Graded Monads). A *graded monad* M on a category $\mathbf{C}$ consists of a family of functors $(M_n\colon \mathbf{C} \to \mathbf{C})_{n<\omega}$, a natural transformation $\eta\colon \mathrm{Id} \to M_0$ (the *unit*) and a family of natural transformations $\mu^{nk}\colon M_n M_k \to M_{n+k}$ for $n, k < \omega$, (the *multiplication*), satisfying the *unit laws*, $\mu^{0n} \cdot \eta M_n = \mathrm{id}_{M_n} = \mu^{n0} \cdot M_n \eta$, for all $n < \omega$, and the *associative law* $\mu^{n,k+m} \cdot M_n \mu^{km} = \mu^{n+k,m} \cdot \mu^{nk} M_m$ for all $k, n, m < \omega$.

Note that it follows that (M_0, η, μ^{00}) is a (plain) monad. For $\mathbf{C} = \mathbf{Set}$, the standard equivalent presentation of monads as algebraic theories carries over to graded monads. Whereas for a monad T, the set TX consists of terms over X modulo equations of the corresponding algebraic theory, for a graded monad $(M_n)_{n<\omega}$, M_nX consists of terms of uniform depth n modulo equations:

▶ **Definition 3.2** (Graded Theories [31])**.** A *graded theory* (Σ, E, d) consists of an algebraic theory, i.e. a (possibly class-sized and infinitary) algebraic signature Σ and a class E of equations, and an assignment d of a *depth* $d(f) < \omega$ to every operation symbol $f \in \Sigma$. This induces a notion of a term *having uniform depth* n: all variables have uniform depth 0, and $f(t_1, \dots, t_n)$ with $d(f) = k$ has uniform depth $n + k$ if all t_i have uniform depth n. (In particular, a constant c has uniform depth n for all $n \geq d(c)$). We require that all equations $t = s$ in E have uniform depth, i.e. that both t and s have uniform depth n for some n. Moreover, we require that for every set X and every $k < \omega$, the class of terms of uniform depth k over variables from X modulo provable equality is small (i.e. in bijection with a set).

Graded theories and graded monads on **Set** are essentially equivalent concepts [31,40]. In particular, a graded theory (Σ, E, d) induces a graded monad M by taking M_nX to be the set of Σ-terms over X of uniform depth n, modulo equality derivable under E.

▶ **Example 3.3.** We recall some examples of graded monads and theories [31].

1. For every endofunctor F on $\mathbf{C}$, the n-fold composition $M_n = F^n$ yields a graded monad with unit $\eta = \mathrm{id}_{\mathrm{Id}}$ and $\mu^{nk} = \mathrm{id}_{F^{n+k}}$.
2. As indicated in the introduction, distributive laws yield graded monads: Suppose that we are given a monad (T, η, μ), an endofunctor F on $\mathbf{C}$ and a distributive law of F over T (a so-called *Kleisli law*), i.e. a natural transformation $\lambda\colon FT \to TF$ such that $\lambda \cdot F\eta = \eta F$ and $\lambda \cdot F\mu = \mu F \cdot T\lambda \cdot \lambda T$. Define natural transformations $\lambda^n\colon F^nT \to TF^n$ inductively by $\lambda^0 = \mathrm{id}_T$ and $\lambda^{n+1} = \lambda^n F \cdot F^n\lambda$. Then we obtain a graded monad with $M_n = TF^n$, unit η, and multiplication $\mu^{nk} = \mu F^{n+1} \cdot T\lambda^n F^k$. The situation is similar for distributive laws of T over F (so-called *Eilenberg-Moore laws*).
3. As a special case of 2., for every monad (T, η, μ) on **Set** and every set $\mathcal{A}$, we obtain a graded monad with $M_nX = T(\mathcal{A}^n \times X)$. Of particular interest to us will be the case where $T = \mathcal{P}_\omega$, which is generated by the algebraic theory of join semilattices (with bottom). The arising graded monad $M_n = \mathcal{P}_\omega(\mathcal{A}^n \times X)$, which is associated with trace equivalence, is generated by the graded theory consisting, at depth 0, of the operations and equations of join semilattices, and additionally a unary operation of depth 1 for each $\sigma \in \mathcal{A}$, subject to (depth-1) equations expressing that these unary operations distribute over joins.

Depth-1 Graded Monads and Theories where operations and equations have depth at most 1 are a particularly convenient case for purposes of building algebras of graded monads; in the following, we elaborate on this condition.

▶ **Definition 3.4** (Depth-1 Graded Theory [31])**.** A graded theory is called *depth*-1 if all its operations and equations have depth at most 1. A graded monad on **Set** is *depth-1* if it can be generated by a depth-1 graded theory.

▶ **Proposition 3.5** (Depth-1 Graded Monads [31])**.** *A graded monad $((M_n), \eta, (\mu^{nk}))$ on* **Set** *is depth-*1 *iff the diagram below is objectwise a coequalizer diagram in* $\mathbf{Set}^{M_0}$ *for all $n < \omega$:*

$$\begin{tikzcd} M_1M_0M_n \arrow[r, shift left, "M_1\mu^{0n}"] \arrow[r, shift right, "\mu^{10}M_n"'] & M_1M_n \arrow[r, "\mu^{1n}"] & M_{1+n}\,. \end{tikzcd} \qquad (1)$$

▶ **Example 3.6.** All graded monads in Example 3.3 are depth 1: for 1., this is easy to see, for 3., it follows from the presentation as a graded theory, and for 2., see [15].

One may use the equivalent property of Proposition 3.5 to define depth-1 graded monads over arbitrary base categories [31]. We show next that depth-1 graded monads may be specified by giving only M_0, M_1, the unit η, and μ^{nk} for $n + k \leq 1$.

▶ **Theorem 3.7.** *Depth-*1 *graded monads are in bijective correspondence with* 6*-tuples* $(M_0, M_1, \eta, \mu^{00}, \mu^{10}, \mu^{01})$ *such that the given data satisfy all applicable instances of the graded monad laws.*

Semantics via Graded Monads. We next recall how graded monads define *graded semantics*:

▶ **Definition 3.8** (Graded semantics [31]). Given a set functor G, a *graded semantics* for G-coalgebras consists of a graded monad $((M_n), \eta, (\mu^{nk}))$ and a natural transformation $\alpha\colon G \to M_1$. The α*-pretrace sequence* $(\gamma^{(n)}\colon X \to M_nX)_{n<\omega}$ for a G-coalgebra $\gamma\colon X \to GX$ is defined by

$$\gamma^{(0)} = (X \xrightarrow{\eta_X} M_0X) \quad \text{and} \quad \gamma^{(n+1)} = (X \xrightarrow{\alpha_X \cdot \gamma} M_1X \xrightarrow{M_1\gamma^{(n)}} M_1M_nX \xrightarrow{\mu^{1n}_X} M_{n+1}X).$$

The α*-trace sequence* T^α_γ is the sequence $(M_n! \cdot \gamma^{(n)}\colon X \to M_n1)_{n<\omega}$.

In **Set**, two states $x \in X$, $y \in Y$ of coalgebras $\gamma\colon X \to GX$ and $\delta\colon Y \to GY$ are α*-trace* (or *graded*) *equivalent* if $M_n! \cdot \gamma^{(n)}(x) = M_n! \cdot \delta^{(n)}(y)$ for all $n < \omega$.

Intuitively, M_nX consists of all length-n *pretraces*, i.e. traces paired with a poststate, and M_n1 consists of all length-n traces, obtained by erasing the poststate. Thus, a graded semantics extracts length-1 pretraces from successor structures. In the following two examples we have $M_1 = G$; however, in general M_1 and G can differ (Section 4).

▶ **Example 3.9.** Recall from Section 2 that a G-coalgebra for the functor $G = \mathcal{P}_\omega(\mathcal{A} \times -)$ is just a finitely branching LTS. We recall two graded semantics that model the extreme ends of the linear time – branching time spectrum [31]; more examples will be given in the next section

1. *Trace equivalence.* For $x, y \in X$ and $w \in \mathcal{A}^*$, we write $x \xrightarrow{w} y$ if y can be reached from x on a path whose labels yield the word w, and $\mathcal{T}(x) = \{w \in \mathcal{A}^* \mid \exists y \in X.\ x \xrightarrow{w} y\}$ denotes the set of *traces* of $x \in X$. States x, y are *trace equivalent* if $\mathcal{T}(x) = \mathcal{T}(y)$. To capture trace semantics of labelled transition systems we consider the graded monad with $M_nX = \mathcal{P}(\mathcal{A}^n \times X)$ (see Example 3.3.3). The natural transformation α is the identity. For a G-coalgebra (X, γ) and $x \in X$ we have that $\gamma^{(n)}(x)$ is the set of pairs (w, y) with $w \in \mathcal{A}^n$ and $x \xrightarrow{w} y$, i.e. pairs of length-n traces and their corresponding poststate. Consequently, the n-th component $M_n! \cdot \gamma^{(n)}$ of the α-trace sequence maps x to the set of its length-n traces. Thus, α-trace equivalence is standard trace equivalence [42].
   Note that the equations presenting the graded monad M_n in Example 3.3.3 bear a striking resemblance to the ones given by van Glabbeek to axiomatize trace equivalence of processes, with the difference that in his axiomatization actions do not distribute over the empty join. In fact, $a.0 = 0$ is clearly not valid for processes under trace equivalence. In the graded setting, this equation just expresses the fact that a trace which ends in a deadlock after n steps cannot be extended to a trace of length $n + 1$.
2. *Bisimilarity.* By the discussion of the final sequence of a functor G (Section 2), the graded monad with $M_nX = G^nX$ (Example 3.3.1), with α being the identity again, captures finite-depth behavioural equivalence, and hence behavioural equivalence when G is finitary. In particular, on finitely branching LTS, α-trace equivalence is bisimilarity in this case.

4 A Spectrum of Graded Monads

We present graded monads for a range of equivalences on the linear time – branching time spectrum as well as probabilistic trace equivalence for generative probabilistic systems (GPS), giving in each case a graded theory and a description of the arising graded monads. Some of our equations bear some similarity to van Glabbeek's axioms for equality of process terms. There are also important differences, however. In particular, some of van Glabbeek's axioms are implications, while ours are purely equational; moreover, van Glabbeek's axioms sometimes nest actions, while we employ only depth-1 equations (which precludes nesting of actions) in order to enable the extraction of characteristic logics later. All graded theories we introduce contain the theory of join semilattices, or in the case of GPS convex algebras, whose operations are assigned depth 0; we mention only the additional operations needed. We use terminology introduced in Example 3.9.

Completed Trace Semantics refines trace semantics by distinguishing whether traces can end in a deadlock. We define a depth-1 graded theory by extending the graded theory for trace semantics (Example 3.3) with a constant depth-1 operation $\star$ denoting deadlock. The induced graded monad has $M_0X = \mathcal{P}_\omega(X)$, $M_1 = \mathcal{P}_\omega(\mathcal{A} \times X + 1)$ (and $M_nX = \mathcal{P}_\omega(\mathcal{A}^n \times X + \mathcal{A}^{<n})$ where $\mathcal{A}^{<n}$ denotes the set of words over $\mathcal{A}$ of length less than n). The natural transformation $\alpha_X : \mathcal{P}_\omega(\mathcal{A} \times X) \to M_1X$ is given by $\alpha(\emptyset) = \{\star\}$ and $\alpha(S) = S \subseteq \mathcal{A} \times X + 1$ for $\emptyset \neq S \subseteq \mathcal{A} \times X$.

Readiness and Failures Semantics refine completed trace semantics by distinguishing which actions are available (readiness) or unavailable (failures) after executing a trace. Formally, given an LTS, seen as a coalgebra $\gamma : X \to \mathcal{P}_\omega(\mathcal{A} \times X)$, we write $I(x) = \mathcal{P}_\omega\pi_1 \cdot \gamma(x) = \pi_1[\gamma(x)]$ (π_1 being the first projection) for the set of actions available at x, the *ready set* of x. A *ready pair* of a state x is a pair $(w, A) \in \mathcal{A}^* \times \mathcal{P}_\omega(\mathcal{A})$ such that there exists z with $x \xrightarrow{w} z$ and $A = I(z)$; a *failure pair* is defined in the same way except that $A \cap I(z) = \emptyset$. Two states are *readiness (failures) equivalent* if they have the same ready (failure) pairs.

We define a depth-1 graded theory by extending the graded theory for trace semantics (Example 3.3) with constant depth-1 operations A for ready (failure) sets $A \subseteq \mathcal{A}$. In case of failures we add a monotonicity condition $A + A \cup B = A \cup B$ on the constant operations for the failure sets. The resulting graded monads both have $M_0X = \mathcal{P}_\omega X$, and moreover $M_1X = \mathcal{P}_\omega(\mathcal{A} \times X + \mathcal{P}_\omega\mathcal{A})$ for readiness and $M_1X = \mathcal{P}^{\downarrow}_\omega(\mathcal{A} \times X + \mathcal{P}_\omega\mathcal{A})$ for failures, where $\mathcal{P}^{\downarrow}_\omega$ is down-closed finite powerset, w.r.t. the discrete order on $\mathcal{A} \times X$ and set inclusion on $\mathcal{P}_\omega\mathcal{A}$. The natural transformation $\alpha_X : \mathcal{P}_\omega(\mathcal{A} \times X) \to M_1X$ is defined by $\alpha_X(S) = S \cup \{\pi_1[S]\}$ for readiness and $\alpha_X(S) = S \cup \{A \subseteq \mathcal{A} \mid A \cap \pi_1[S] = \emptyset\}$ for failures semantics.

Ready Trace and Failure Trace Semantics refine readiness and failures semantics, respectively, by distinguishing which actions are available (ready trace) or unavailable (failure trace) at each step of the trace. Formally, a *ready trace* of a state x is a sequence $A_0a_1A_1 \dots a_nA_n \in (\mathcal{P}_\omega\mathcal{A} \times \mathcal{A})^* \times \mathcal{P}_\omega\mathcal{A}$ such that there exist transitions $x = x_0 \xrightarrow{a_1} x_1 \dots \xrightarrow{a_n} x_n$ where each x_i has ready set $I(x_i) = A_i$. A *failure trace* has the same shape but we require that each A_i is a *failure set* of x_i, i.e. $I(x_i) \cap A_i = \emptyset$. States are *ready (failure) trace equivalent* if they have the same ready (failure) traces.

For ready traces, we define a depth-1 graded theory with depth-1 operations $\langle A, \sigma\rangle$ for $\sigma \in \mathcal{A}$, $A \subseteq \mathcal{A}$ and a depth-1 constant $\star$, denoting deadlock, and equations $\langle A, \sigma\rangle(\sum_{j\in J} x_j) = \sum_{j\in J}\langle A, \sigma\rangle(x_j)$. The resulting graded monad is simply the graded monad capturing completed trace semantics for labelled transition systems where the set of actions is changed from $\mathcal{A}$ to $\mathcal{P}_\omega\mathcal{A} \times \mathcal{A}$. For failure traces, we additionally impose the equation $\langle A, \sigma\rangle(x) + \langle A \cup B, \sigma\rangle(x) = \langle A \cup B, \sigma\rangle(x)$, which in the set-based description of the graded monad corresponds to downward closure of failure sets.

The resulting graded monads both have $M_0X = \mathcal{P}_\omega X$; for ready traces, $M_1X = \mathcal{P}_\omega((\mathcal{P}_\omega\mathcal{A} \times \mathcal{A}) \times X + 1)$ and for failure traces, $M_1X = \mathcal{P}^{\downarrow}_\omega((\mathcal{P}_\omega\mathcal{A} \times \mathcal{A}) \times X + 1)$, where $\mathcal{P}^{\downarrow}_\omega$ is down-closed finite powerset, w.r.t. the order imposed by the above equation.

For ready trace semantics we define the natural transformation $\alpha_X : \mathcal{P}_\omega(\mathcal{A} \times X) \to M_1X$ by $\alpha_X(\emptyset) = \{\star\}$ and $\alpha_X(S) = \{((\pi_1[S], \sigma), x) \mid (\sigma, x) \in S\})$ for $S \neq \emptyset$. For failure traces we define $\alpha_X(\emptyset) = \{\star\}$ and $\alpha(S) = \{((A, \sigma), x) \mid (\sigma, x) \in S, A \cap \pi_1[S] = \emptyset\}$ for $S \neq \emptyset$; note that in the latter case, $\alpha(S)$ is closed under decreasing failure sets.

Simulation Equivalence declares two states to be equivalent if they simulate each other in the standard sense. We define a depth-1 graded theory with the same signature as for trace equivalence but instead of join preservation require only that each σ is monotone, i.e. $\sigma(x + y) + \sigma(x) = \sigma(x + y)$. The arising graded monad M_n is equivalently described as follows. We define the sets M_nX inductively, along with an ordering on M_nX. We take $M_0X = \mathcal{P}_\omega X$, ordered by set inclusion. We then order the elements of $\mathcal{A} \times M_nX$ by the product ordering of the discrete order on $\mathcal{A}$ and the given ordering on M_nX, and take $M_{n+1}X$ to be the set of downclosed subsets of $\mathcal{A} \times M_nX$, denoted $\mathcal{P}^{\downarrow}_\omega(\mathcal{A} \times M_nX)$, ordered by set inclusion. The natural transformation $\alpha_X : \mathcal{P}(\mathcal{A} \times X) \to \mathcal{P}^{\downarrow}_\omega(\mathcal{A} \times \mathcal{P}_\omega(X))$ is defined by $\alpha_X(S) = {\downarrow}\{(s, \{x\}) \mid (s, x) \in S\}$, where $\downarrow$ denotes downclosure.

Ready Simulation Equivalence refines simulation equivalence by requiring additionally that related states have the same ready set. States x and y are *ready similar* if they are related by some ready simulation, and ready simulation equivalent if there are mutually ready similar. The depth-1 graded theory combines the signature for ready traces with the equations for simulation, i.e. only requires the operations $\langle A, \sigma\rangle$ to be monotone.

Probabilistic Trace Equivalence is the standard trace semantics for generative probabilistic systems (GPS), equivalently, coalgebras for the functor $\mathcal{D}(\mathcal{A} \times \mathrm{Id})$ where $\mathcal{D}$ is the monad of finitary distributions (Example 2.1). Probabilistic trace equivalence is captured by the graded monad $M_nX = \mathcal{D}(\mathcal{A}^n \times X)$, as described in Example 3.3.2. The corresponding graded theory arises by replacing the join-semilattice structure featuring in the above graded theory for trace equivalence by the one of *convex algebras*, i.e. the algebras for the monad $\mathcal{D}$. Recall [13, 14] that a convex algebra is a set X equipped with finite convex sum operations: For every $p \in [0, 1]$ there is a binary operation $\boxplus_p$ on X, and these operations satisfy the equations $x \boxplus_p x = x, x \boxplus_p y = y \boxplus_{1-p} x, x \boxplus_0 y = y, x \boxplus_p (y \boxplus_q z) = (x \boxplus_{p/r} y) \boxplus_r z$, where $p, q \in [0, 1]$, $x, y, z \in X$, and $r = (p+(1-p)q) > 0$ (i.e. $p+q > 0$) in the last equation [21]. Again, we have depth-1 operations σ for action $\sigma \in \mathcal{A}$, now satisfying the equations $\sigma(x \boxplus_p y) = \sigma(x) \boxplus_p \sigma(y)$.

5 Graded Logics

Our next goal is to extract *characteristic logics* from graded monads in a systematic way, with *characterizing* meaning that states are logically indistinguishable iff they are equivalent under the semantics at hand. We will refer to these logics as *graded logics*; the implication from graded equivalence to logical indistinguishability is called *invariance*, and the converse implication *expressiveness*. E.g. standard modal logic with the full set of Boolean connectives is invariant under bisimilarity, and the corresponding expressiveness result is known as the *Hennessy-Milner theorem*. This result has been lifted to coalgebraic generality early on, giving rise to the *coalgebraic Hennessy-Milner theorem* [33, 36]. In previous work [31], we have related graded semantics to modal logics extracted from the graded monad in the envisaged fashion. These logics are invariant by construction; the main new result we present here is a generic *expressiveness* criterion, to be discussed in Section 6. The key ingredient in this criterion are *canonical* graded algebras, which we newly introduce here, providing a recursive-evaluation style reformulation of the semantics of graded logics.

A further key issue in characteristic modal logics is the choice of propositional operators; e.g. notice that when $\Diamond_\sigma$ denotes the usual Hennessy-Milner style diamond operator for an action σ, the formula $\Diamond_\sigma\top \wedge \Diamond_\tau\top$ is invariant under trace equivalence (i.e. the corresponding property is closed under under trace equivalence) but the formula $\Diamond_\sigma(\Diamond_\sigma\top \wedge \Diamond_\tau\top)$, built from the former by simply prefixing with $\Diamond_\sigma$, is not, the problem being precisely the use of conjunction. While in our original setup, propositional operators were kept implicit, that is, incorporated into the set of modalities, we provide an explicit treatment of propositional operators in the present paper. Besides adding transparency to the syntax and semantics, having first-class propositional operators will be a prerequisite for the formulation of the expressiveness theorem.

Coalgebraic Modal Logic. To provide context, we briefly recall the setup of *coalgebraic modal logic* [33, 36]. Let 2 denote the set $\{\bot, \top\}$ of Boolean truth values; we think of the set 2^X of maps $X \to 2$ as the set of predicates on X. Coalgebraic logic in general abstracts systems as coalgebras for a functor G, like we do here; fixes a set Λ of *modalities* (unary for the sake of readability); and then interprets a modality $L \in \Lambda$ by the choice of a *predicate lifting*, i.e. a natural transformation

$$[\![L]\!]_X \colon 2^X \to 2^{GX}.$$

By the Yoneda lemma, such natural transformations are in bijective correspondence with maps $G2 \to 2$ [36], which we shall also denote as $[\![L]\!]$. In the latter formulation, the recursive clause defining the interpretation $[\![L\phi]\!] \colon X \to 2$, for a modal formula ϕ, as a state predicate in a G-coalgebra $\gamma\colon X \to GX$ is then

$$[\![L\phi]\!] = (X \xrightarrow{\gamma} GX \xrightarrow{G[\![\phi]\!]} G2 \xrightarrow{[\![L]\!]} 2). \tag{2}$$

E.g. taking $G = \mathcal{P}_\omega(\mathcal{A} \times -)$ (for labelled transition systems), we obtain the standard semantics of the Hennessy-Milner diamond modality $\Diamond_\sigma$ for $\sigma \in \mathcal{A}$ via the predicate lifting

$$[\![\Diamond_\sigma]\!]_X(f) = \{B \in \mathcal{P}_\omega(\mathcal{A} \times X) \mid \exists x.\, (\sigma, x) \in B \wedge f(x) = \top\} \qquad (\text{for } f\colon X \to 2).$$

It is easy to see that *coalgebraic modal logic*, which combines coalgebraic modalities with the full set of Boolean connectives, is invariant under finite-depth behavioural equivalence (Section 2). Generalizing the classical Hennessy-Milner theorem [19], the *coalgebraic Hennessy-Milner theorem* [33, 36] shows that conversely, coalgebraic modal logic *characterizes* behavioural equivalence, i.e. logical indistinguishability implies behavioural equivalence, provided that G is finitary (implying coincidence of behavioural equivalence and finite-depth behavioural equivalence) and Λ is *separating*, i.e. for every finite set X, the set

$$\Lambda(2^X) = \{[\![L]\!](f) \mid f \in 2^X\}$$

of maps $GX \to 2$ is jointly injective.

We proceed to introduce the syntax and semantics of graded logics.

Syntax. We parametrize the syntax of *graded logics* over

- a set Θ of *truth constants*,
- a set $\mathcal{O}$ of *propositional operators* with assigned finite arities, and
- a set Λ of *modalities* with assigned arities.

For readability, we will restrict the technical exposition to unary modalities; the treatment of higher arities requires no more than additional indexing (and we will use 0-ary modalities in the examples). E.g. standard Hennessy-Milner logic is given by $\Lambda = \{\Diamond_\sigma \mid \sigma \in \mathcal{A}\}$ and $\mathcal{O}$ containing all Boolean connectives. Other logics will be determined by additional or different modalities, and often by fewer propositional operators. Formulae of the logic are restricted to have uniform depth, where propositional operators have depth 0 and modalities have depth 1; a somewhat particular feature is that truth constants can have top-level occurrences only in depth-0 formulae. That is, formulae $\phi, \phi_1, \dots$ of depth 0 are given by the grammar

$$\phi ::= p(\phi_1, \dots, \phi_k) \mid c \qquad (p \in \mathcal{O} \text{ } k\text{-ary}, c \in \Theta),$$

and formulae ϕ of depth $n+1$ by

$$\phi ::= p(\phi_1, \dots, \phi_k) \mid L\psi \qquad (p \in \mathcal{O} \text{ } k\text{-ary}, L \in \Lambda)$$

where $\phi_1, \dots, \phi_n$ range over formulae of depth $n+1$ and ψ over formulae of depth n.

Semantics. The semantics of graded logics is parametrized over the choice of *a functor G, a depth-1 graded monad $M = ((M_n)_{n<\omega}, \eta, (\mu^{nk})_{n,k<\omega})$, and a graded semantics $\alpha\colon G \to M_1$, which we fix for the remainder of the paper.* It was originally given by translating formulae into *graded algebras* and then defining formula evaluation by the universal property of $(M_n 1)$ as a free graded algebra [31]; here, we reformulate the semantics in a more standard style by recursive clauses, using canonical graded algebras. In general, the notion of graded algebra is defined as follows [31].

▶ **Definition 5.1** (Graded algebras)**.** Let $n < \omega$. A *(graded) M_n-algebra* $A = ((A_k)_{k \le n}, (a^{mk})_{m+k \le n})$ consists of carrier sets A_k and structure maps

$$a^{mk}\colon M_m A_k \to A_{m+k}$$

satisfying the laws

$$\begin{tikzcd} A_k \arrow[r, "\eta_{A_k}"] \arrow[dr, equal] & M_0 A_k \arrow[d, "a^{0k}"] & M_m M_r A_k \arrow[r, "M_m a^{rk}"] \arrow[d, "\mu^{mr}_{A_k}"'] & M_m A_{r+k} \arrow[d, "a^{m,r+k}"] \\ & A_k & M_{m+r} A_k \arrow[r, "a^{m+r,k}"] & A_{m+r+k} \end{tikzcd} \tag{3}$$

for all $k \le n$ (left) and all m, r, k such that $m + r + k \le n$ (right), respectively. An *M_n-morphism* f from A to an M_n-algebra $B = ((B_k)_{k \le n}, (b^{mk})_{m+k \le n})$ consists of maps $f_k\colon A_k \to B_k$, $k \le n$, such that $f_{m+k} \cdot a^{mk} = b^{mk} \cdot M_m f_k$ for all m, k such that $m + k \le n$.

We view the carrier A_k of an M_n-algebra as the set of algebra elements that have already absorbed operations up to depth k. As in the case of plain monads, we can equivalently describe graded algebras in terms of graded theories: If M is generated by a graded theory $\mathbb{T} = (\Sigma, E, d)$, then an M_n-algebra interprets each operation $f \in \Sigma$ of arity r and depth $d(f) = m$ by maps $f^A_k\colon A^r_k \to A_{m+k}$ for all k such that $m + k \le n$; this gives rise to an inductively defined interpretation of terms (specifically, given a valuation of variables in A_m, terms of uniform depth k receive values in A_{k+m}, for $k + m \le n$), and subsequently to the expected notion of satisfaction of equations.

While in general, graded algebras are monolithic objects, for depth-1 graded monads we can construct them in a modular fashion from M_1-algebras [31]; we thus restrict attention to M_0- and M_1-algebras in the following. We note that an M_0-algebra is just an Eilenberg-Moore algebra for the monad M_0. An M_1-Algebra A consists of M_0-algebras $(A_0, a^{00}\colon M_0A_0 \to A_0)$ and $(A_1, a^{01}\colon M_0A_1 \to A_1)$, and a *main structure map* $a^{10}\colon M_1A_0 \to A_1$ satisfying two instances of the right-hand diagram in (3), one of which says that a^{10} is a morphism of M_0-algebras (*homomorphy*), and the other that the diagram

$$\begin{tikzcd} M_1M_0A_0 \arrow[r, shift left, "\mu^{10}"] \arrow[r, shift right, "M_1a^{00}"'] & M_1A_0 \arrow[r, "a^{10}"] & A_1, \end{tikzcd} \tag{4}$$

which by the laws of graded monads consists of M_0-algebra morphisms, commutes (*coequalization*). We will often refer to an M_1-algebra by just its main structure map.

We will use M_1-algebras as interpretations of the modalities in graded logics, generalizing the previously recalled interpretation of modalities as maps $G2 \to 2$ in branching-time coalgebraic modal logic. We fix an M_0-algebra Ω of *truth values*, with structure map $o\colon M_0\Omega \to \Omega$ (e.g. for $G = \mathcal{P}_\omega$, Ω is a join semilattice). Powers Ω^n of Ω are again M_0-algebras. A modality $L \in \Lambda$ is interpreted as an M_1-algebra $A = [\![L]\!]$ with carriers $A_0 = A_1 = \Omega$ and $a^{01} = a^{00} = o$. Such an M_1-algebra is thus specified by its main structure map $a^{10}\colon M_1\Omega \to \Omega$ alone, so following the convention indicated above we often write $[\![L]\!]$ for just this map. The evaluation of modalities is defined using canonical M_1-algebras:

▶ **Definition 5.2** (Canonical algebras). The 0-*part* of an M_1-algebra A is the M_0-algebra (A_0, a^{00}). Taking 0-parts defines a functor U_0 from M_1-algebras to M_0-algebras. An M_1-algebra is *canonical* if it is free, w.r.t. U_0, over its 0-part. For A canonical and a modality $L \in \Lambda$, we denote the unique morphism $A_1 \to \Omega$ extending an M_0-morphism $f\colon A_0 \to \Omega$ to an M_1-morphism $A \to [\![L]\!]$ by $[\![L]\!](f)$, i.e. $[\![L]\!](f)$ is the unique M_0-morphism such that the following equation holds:

$$(M_1A_0 \xrightarrow{M_1f} M_1\Omega \xrightarrow{[\![L]\!]} \Omega) = (M_1A_0 \xrightarrow{a^{10}} A_1 \xrightarrow{[\![L]\!](f)} \Omega). \tag{5}$$

▶ **Lemma 5.3.** *An M_1-algebra A is canonical iff* (4) *is a (reflexive) coequalizer diagram in the category of M_0-algebras.*

By the above lemma, we obtain a key example of canonical M_1-algebras:

▶ **Corollary 5.4.** *If M is a depth-1 graded monad, then for every n and every set X, the M_1-algebra with carriers $M_nX, M_{n+1}X$ and multiplication as algebra structure is canonical.*

Further, we interpret truth constants $c \in \Theta$ as elements of Ω, understood as maps $\hat{c}\colon 1 \to \Omega$, and k-ary propositional operators $p \in \mathcal{O}$ as M_0-homomorphisms $[\![p]\!]\colon \Omega^k \to \Omega$. In our examples on the linear time – branching time spectrum, M_0 is either the identity or, most of the time, the finite powerset monad. In the former case, all truth functions are M_0-morphisms. In the latter case, the M_0-morphisms $\Omega^k \to \Omega$ are the join-continuous functions; in the standard case where $\Omega = 2$ is the set of Boolean truth values, such functions f have the form $f(x_1, \dots, x_k) = x_{i_1} \vee \dots \vee x_{i_l}$, where $i_1, \dots, i_l \in \{1, \dots, k\}$. We will see one case where M_0 is the distribution monad; then M_0-morphisms are affine maps.

The semantics of a formula ϕ in graded logic is defined recursively as an M_0-morphism $[\![\phi]\!]\colon (M_n1, \mu_1^{0n}) \to (\Omega, o)$ by

$$[\![c]\!] = (M_01 \xrightarrow{M_0\hat{c}} M_0\Omega \xrightarrow{o} \Omega) \quad [\![p(\phi_1, \dots, \phi_k)]\!] = [\![p]\!] \cdot \langle [\![\phi_1]\!], \dots, [\![\phi_k]\!] \rangle \quad [\![L\phi]\!] = [\![L]\!]([\![\phi]\!]).$$

The evaluation of ϕ in a coalgebra $\gamma\colon X \to GX$ is then given by composing with the trace sequence, i.e. as

$$X \xrightarrow{M_n! \cdot \gamma^{(n)}} M_n 1 \xrightarrow{[\![\phi]\!]} \Omega. \tag{6}$$

In particular, graded logics are, by construction, invariant under the graded semantics.

▶ **Example 5.5** (Graded logics). We recall the two most basic examples, fixing $\Omega = 2$ in both cases, and $\top$ as the only truth constant:

1. *Finite-depth behavioural equivalence:* Recall that the graded monad $M_n X = G^n X$ captures finite-depth behavioural equivalence on G-coalgebras. Since M_0 is the identity monad, M_0-algebras are just sets. Thus, every function $2^k \to 2$ is an M_0-morphism, so we can use all Boolean operators as propositional operators. Moreover, M_1-algebras are just maps $a^{10}\colon GA_0 \to A_1$. Such an M_1-algebra is canonical iff a^{10} is an isomorphism, and modalities are interpreted as M_1-algebras $G2 \to 2$, with the evaluation according to (5) and (6) corresponding precisely to the semantics of modalities in coalgebraic logic (2). Summing up, we obtain precisely coalgebraic modal logic as summarized above in this case. In our running example $G = \mathcal{P}_\omega(\mathcal{A} \times (-))$, we take modalities $\Diamond_\sigma$ as above, with $[\![\Diamond_\sigma]\!]\colon \mathcal{P}_\omega(\mathcal{A} \times 2) \to 2$ defined by $[\![\Diamond_\sigma]\!](S) = \top$ iff $(\sigma, \top) \in S$, obtaining precisely classical Hennessy-Milner logic [19].
2. *Trace equivalence:* Recall that the trace semantics of labelled transition systems with actions in $\mathcal{A}$ is modelled by the graded monad $M_n X = \mathcal{P}_\omega(\mathcal{A}^n \times X)$. As indicated above, in this case we can use disjunction as a propositional operator since $M_0 = \mathcal{P}_\omega$. Since the graded theory for M_n specifies for each $\sigma \in \mathcal{A}$ a unary depth-1 operation that distributes over joins, we find that the maps $[\![\Diamond_\sigma]\!]$ from the previous example (unlike their duals $\Box_\sigma$) induce M_1-algebras also in this case, so we obtain a graded trace logic featuring precisely diamonds and disjunction, as expected.

We defer the discussion of further examples, including ones where $\Omega = [0, 1]$, to the next section, where we will simultaneously illustrate the generic expressiveness result (Example 6.5).

▶ Remark 5.6. One important class of examples where the above approach to characteristic logics will *not* work without substantial further development are simulation-like equivalences, whose characteristic logics need conjunction [42]. Conjunction is not an M_0-morphism for the corresponding graded monads identified in Section 4, which both have $M_0 = \mathcal{P}_\omega$. A related and maybe more fundamental observation is that formula evaluation is not M_0-morphic in the presence of conjunction; e.g. over simulation equivalence, the evaluation map $M_1 1 = \mathcal{P}_\omega^{\downarrow}(\mathcal{A} \times \mathcal{P}_\omega(1)) \to 2$ of the formula $\Diamond_\sigma \top \wedge \Diamond_\tau \top$ fails to be join-continuous for distinct $\sigma, \tau \in \mathcal{A}$. We leave the extension of our logical framework to such cases to future work, expecting a solution in elaborating the theory of graded monads, theories, and algebras over the category of partially ordered sets, where simulations live more naturally (e.g. [24]).

6 Expressiveness

We now present our main result, an expressiveness criterion for graded logics, which states that a graded logic characterizes the given graded semantics if it has enough modalities propositional operators, and truth constants. Both the criterion and its proof now fall into place naturally and easily, owing to the groundwork laid in the previous section, in particular the reformulation of the semantics in terms of canonical algebras:

▶ **Definition 6.1.** We say that a graded logic with set Ω of truth values and sets Θ, $\mathcal{O}$, Λ of truth constants, propositional operators, and modalities, respectively, is

1. *depth-*0 *separating* if the family of maps $[\![c]\!]\colon M_0 1 \to \Omega$, for truth constants $c \in \Theta$, is jointly injective; and
2. *depth-*1 *separating* if, whenever A is a canonical M_1-algebra and $\mathfrak{A}$ is a jointly injective set of M_0-homomorphisms $A_0 \to \Omega$ that is closed under the propositional operators in $\mathcal{O}$ (in the sense that $[\![p]\!] \cdot \langle f_1, \dots, f_k\rangle \in \mathfrak{A}$ for $f_1, \dots, f_k \in \mathfrak{A}$ and k-ary $p \in \mathcal{O}$), then the set $\Lambda(\mathfrak{A}) := \{[\![L]\!](f)\colon A_1 \to \Omega \mid L \in \Lambda, f \in \mathfrak{A}\}$ of maps is jointly injective.

▶ **Theorem 6.2** (Expressiveness). *If a graded logic is both depth-*0 *separating and depth-*1 *separating, then it is expressive.*

▶ **Example 6.3** (Logics for bisimilarity). We note first that the existing coalgebraic Hennessy-Milner theorem, for branching time equivalences and coalgebraic modal logic with full Boolean base over a finitary functor G [33,36], as recalled in Section 5, is a special case of Theorem 6.2: We have already seen in Example 5.5 that coalgebraic modal logic in the above sense is an instance of our framework for the graded monad $M_n X = G^n X$. Since $M_0 = \mathrm{id}$ in this case, depth-0 separation is vacuous. As indicated in Example 5.5, canonical M_1-algebras are w.l.o.g. of the form $\mathrm{id}\colon GX \to GX$, where for purposes of proving depth-1 separation, we can restrict to finite X since G is finitary. Then, a set $\mathfrak{A}$ as in Definition 6.1 is already the whole powerset 2^X, so depth-1 separation is exactly the previous notion of separation.

A well-known particular case is probabilistic bisimilarity on Markov chains, for which an expressive logic needs only probabilistic modalities $\Diamond_p$ "with probability at least p" and conjunction [12]. This result (later extended to more complex composite functors [32]) is also easily recovered as an instance of Theorem 6.2, using the same standard lemma from measure theory as in *op. cit.*, which states that measures are uniquely determined by their values on a generating set of the underlying σ-algebra that is closed under finite intersections (corresponding to the set $\mathfrak{A}$ from Definition 6.1 being closed under conjunction).

▶ Remark 6.4. For behavioural equivalence, i.e. $M_n X = G^n X$ as in the above example, the inductive proof of our expressiveness theorem essentially instantiates to Pattinson's proof of the coalgebraic Hennessy-Milner theorem by induction over the terminal sequence [33]. One should note that although the coalgebraic Hennessy-Milner theorem can be shown to hold for larger cardinal bounds on the branching by means of a direct quotienting construction [36], the terminal sequence argument goes beyond finite branching only in corner cases.

▶ **Example 6.5** (Expressive graded logics on the linear time – branching time spectrum). We next extract graded logics from some of the graded monads for the linear time – branching time spectrum introduced in Section 4, and show how in each case, expressiveness is an instance of Theorem 6.2. Bisimilarity is already covered by the previous example. Depth-0 separation is almost always trivial and not mentioned further. Unless mentioned otherwise, all logics have disjunction, enabled by M_0 being powerset as discussed in the previous section. Most of the time, the logics are essentially already given by van Glabbeek (with the exception that we show that one can add disjunction) [42]; the emphasis is entirely on uniformization.

1. *Trace equivalence:* As seen in Example 5.5, the graded logic for trace equivalence features (disjunction and) diamond modalities $\Diamond_\sigma$ indexed over actions $\sigma \in \mathcal{A}$. The ensuing proof of depth-1 separation uses canonicity of a given M_1-algebra A only to obtain that the structure map a^{10} is surjective. The other key point is that a jointly injective collection $\mathfrak{A}$ of M_0-homomorphisms $A_0 \to 2$, i.e. join preserving maps, has the stronger separation property that whenever $x \not\leq y$ then there exists $f \in \mathfrak{A}$ such that $f(x) = \top$ and $f(y) = \bot$.

2. Graded logics for completed traces, readiness, failures, ready traces, and failure traces are developed from the above by adding constants or additionally indexing modalities over sets of actions, with only little change to the proofs of depth-1 separation. For completed trace equivalence, we just add a 0-ary modality $\star$ indicating deadlock. For ready trace equivalence, we index the diamond modalities $\Diamond_\sigma$ with sets $I \subseteq \mathcal{A}$; formulae $\Diamond_{\sigma,I}\phi$ are then read 'the current ready set is I, and there is a σ-successor satisfying ϕ'. For failure trace equivalence we proceed in the same way but read the index I as 'I is a failure set at the current state'. For readiness equivalence and failures equivalence, we keep the modalities $\Diamond_\sigma$ unchanged from trace equivalence and instead introduce 0-ary modalities r_I indicating that I is the ready set or a failure set, respectively, at the current state, thus ensuring that formulae do not continue after postulating a ready set.

▶ **Example 6.6** (Probabilistic traces)**.** We have recalled in Section 4 that probabilistic trace equivalence of generative probabilistic transition systems can be captured as a graded semantics using the graded monad $M_nX = \mathcal{D}(\mathcal{A}^n \times X)$, with M_0-algebras being convex algebras. In earlier work [31] we have noted that a logic over the set $\Omega = [0,1]$ of truth values (with the usual convex algebra structure) featuring rational truth constants, affine combinations as propositional operators (as indicated in Section 5), and modal operators $\langle\sigma\rangle$, interpreted by M_1-algebras $[\![\langle\sigma\rangle]\!]\colon M_1[0,1] \to [0,1]$ defined by $[\![\langle\sigma\rangle]\!](\mu) = \sum_{r\in[0,1]} r\mu(\sigma,r)$ is invariant under probabilistic trace equivalence. By our expressiveness criterion, we recover the result that this logic is expressive for probabilistic trace semantics (see e.g. [2]).

7 Conclusion and Future Work

We have provided graded monads modelling a range of process equivalences on the linear time – branching time spectrum, presented in terms of carefully designed graded algebraic theories. From these graded monads, we have extracted characteristic modal logics for the respective equivalences systematically, following a paradigm of graded logics that grows out of a natural notion of graded algebra. Our main technical results concern the further development of the general framework for graded logics; in particular, we have introduced a first-class notion of propositional operator, and we have established a criterion for *expressiveness* of graded logics that simultaneously takes into account the expressive power of the modalities and that of the propositional base. (An open question that remains is whether an expressive logic always exists, as it does in the branching-time setting [36].) Instances of this result include, for instance, the coalgebraic Hennessy-Milner theorem [33, 36], Desharnais et al.'s expressiveness result for probabilistic modal logic with only conjunction [12], and expressiveness for various logics for trace-like equivalences on non-deterministic and probabilistic systems. The emphasis in the examples has been on well-researched equivalences and logics for the basic case of labelled transition systems, aimed at demonstrating the versatility of graded monads and graded logics along the axis of granularity of system equivalence. The framework as a whole is however parametric also over the branching type of systems and in fact over the base category determining the structure of state spaces; an important direction for future research is therefore to capture (possibly new) equivalences and extract expressive logics on other system types such as probabilistic systems (we have already seen probabilistic trace equivalence as an instance; see [4] for a comparison of some equivalences on probabilistic automata, which combine probabilities and non-determinism) and nominal systems, e.g. nominal automata [3, 37]. Moreover, we plan to extend the framework of graded logics to cover also temporal logics, using graded algebras of unbounded depth.

References

1 Jirí Adámek and Václav Koubek. Remarks on Fixed Points of Functors. In *Fundamentals of Computation Theory, FCT 1977*, LNCS, pages 199–205. Springer, 1977. doi:10.1007/3-540-08442-8_86.

2 Marco Bernardo and Stefania Botta. A survey of modal logics characterising behavioural equivalences for non-deterministic and stochastic systems. *Math. Struct. Comput. Sci.*, 18:29–55, 2008.

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

4 Filippo Bonchi, Alexandra Silva, and Ana Sokolova. The Power of Convex Algebras. In *Concurrency Theory, CONCUR 2017*, volume 85 of *LIPIcs*, pages 23:1–23:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

5 Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. The Theory of Traces for Systems with Nondeterminism and Probability. In *Logic in Computer Science, LICS 2019*. IEEE, 2019.

6 Marcello Bonsangue, Stefan Milius, and Alexandra Silva. Sound and complete axiomatizations of coalgebraic language equivalence. *ACM Trans. Comput. Log.*, 14, 2013.

7 Corina Cîrstea. Maximal traces and path-based coalgebraic temporal logics. *Theoret. Comput. Sci.*, 412(38):5025–5042, 2011. doi:10.1016/j.tcs.2011.04.025.

8 Corina Cîrstea. A Coalgebraic Approach to Linear-Time Logics. In *Foundations of Software Science and Computation Structures, FoSSaCS 2014*, volume 8412 of *LNCS*, pages 426–440. Springer, 2014.

9 Corina Cîrstea. Canonical Coalgebraic Linear Time Logics. In *Algebra and Coalgebra in Computer Science, CALCO 2015*, Leibniz International Proceedings in Informatics, 2015.

10 Corina Cîrstea. From Branching to Linear Time, Coalgebraically. *Fund. Inform.*, 150(3-4):379–406, 2017. doi:10.3233/FI-2017-1474.

11 Yuxin Deng, Rob van Glabbeek, Matthew Hennessy, and Carroll Morgan. Characterising Testing Preorders for Finite Probabilistic Processes. *Log. Meth. Comput. Sci.*, 4(4), 2008. doi:10.2168/LMCS-4(4:4)2008.

12 Josee Desharnais, Abbas Edalat, and Prakash Panangaden. A Logical Characterization of Bisimulation for Labeled Markov Processes. In *Logic in Computer Science, LICS 1998*, pages 478–487. IEEE Computer Society, 1998.

13 Ernst-Erich Doberkat. Eilenberg-Moore algebras for stochastic relations. *Inf. Comput.*, 204(12):1756–1781, 2006. doi:10.1016/j.ic.2006.09.001.

14 Ernst-Erich Doberkat. Erratum and Addendum: Eilenberg-Moore algebras for stochastic relations. *Inf. Comput.*, 206(12):1476–1484, 2008. doi:10.1016/j.ic.2008.08.002.

15 Ulrich Dorsch, Stefan Milius, and Lutz Schröder. Graded Monads and Graded Logics for the Linear Time – Branching Time Spectrum, 2019. arXiv:1812.01317.

16 Laurent Doyen, Thomas Henzinger, and Jean-François Raskin. Equivalence of Labeled Markov Chains. *Int. J. Found. Comput. Sci.*, 19(3):549–563, 2008. doi:10.1142/S0129054108005814.

17 Uli Fahrenberg and Axel Legay. A Linear-Time-Branching-Time Spectrum of Behavioral Specification Theories. In *Theory and Practice of Computer Science, SOFSEM 2017*, volume 10139 of *LNCS*, pages 49–61. Springer, 2017. doi:10.1007/978-3-319-51963-0.

18 Ichiro Hasuo, Bart Jacobs, and Ana Sokolova. Generic Trace Semantics via Coinduction. *Log. Meth. Comput. Sci.*, 3, 2007. doi:10.2168/LMCS-3(4:11)2007.

19 M. Hennessy and R. Milner. Algebraic Laws for Non-Determinism and Concurrency. *J. ACM*, 32:137–161, 1985.

20 Bart Jacobs. Trace Semantics for Coalgebras. In J. Adámek and S. Milius, editors, *Coalgebraic Methods in Computer Science, CMCS 2004*, volume 106 of *ENTCS*, pages 167–184. Elsevier, 2004. doi:10.1016/j.entcs.2004.02.031.

21 Bart Jacobs. Convexity, Duality and Effects. In C.S. Calude and V. Sassone, editors, *Proc. TCS 2010*, volume 323 of *IFIP AICT*, pages 1–19, 2010.

22 Bart Jacobs, Paul B. Levy, and Jurriaan Rot. Steps and Traces. In Corina Cîrstea, editor, *Proc. CMCS 2018*, volume 11202 of *LNCS*, pages 122–143. Springer, 2018.

23 Bart Jacobs, Alexandra Silva, and Ana Sokolova. Trace Semantics via Determinization. In *Coalgebraic Methods in Computer Science, CMCS 2012*, volume 7399 of *LNCS*, pages 109–129. Springer, 2012. doi:10.1007/978-3-642-32784-1.

24 Krzysztof Kapulkin, Alexander Kurz, and Jiri Velebil. Expressiveness of Positive Coalgebraic Logic. In *Advances in Modal Logic, AiML 2012*, pages 368–385. College Publications, 2012.

25 Henning Kerstan and Barbara König. Coalgebraic Trace Semantics for Continuous Probabilistic Transition Systems. *Log. Meth. Comput. Sci.*, 9(4), 2013. doi:10.2168/LMCS-9(4:16)2013.

26 Christian Kissig and Alexander Kurz. Generic Trace Logics. arXiv preprint 1103.3239, 2011.

27 Bartek Klin. Structural Operational Semantics for Weighted Transition Systems. In *Semantics and Algebraic Specification*, volume 5700 of *LNCS*, pages 121–139. Springer, 2009.

28 Bartek Klin and Juriaan Rot. Coalgebraic trace semantics via forgetful logics. In *Foundations of Software Science and Computation Structures, FoSSaCS 2015*, volume 9034 of *LNCS*, pages 151–166. Springer, 2015.

29 K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94:1–28, 1991.

30 Saunders MacLane. *Categories for the working mathematician.* Springer, 2nd edition, 1998.

31 Stefan Milius, Dirk Pattinson, and Lutz Schröder. Generic Trace Semantics and Graded Monads. In *Algebra and Coalgebra in Computer Science, CALCO 2015*, Leibniz International Proceedings in Informatics, 2015.

32 Lawrence Moss and Ignacio Viglizzo. Final coalgebras for functors on measurable spaces. *Inf. Comput.*, 204(4):610–636, 2006. doi:10.1016/j.ic.2005.04.006.

33 D. Pattinson. Expressive Logics for Coalgebras via Terminal Sequence Induction. *Notre Dame J. Formal Log.*, 45:19–33, 2004.

34 Benjamin Pierce. *Basic category theory for computer scientists.* MIT Press, 1991.

35 J. Rutten. Universal Coalgebra: A Theory of Systems. *Theoret. Comput. Sci.*, 249:3–80, 2000.

36 Lutz Schröder. Expressivity of Coalgebraic Modal Logic: The Limits and Beyond. *Theoret. Comput. Sci.*, 390:230–247, 2008.

37 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.

38 Roberto Segala and Nancy Lynch. Probabilistic Simulations for Probabilistic Processes. In *Concurrency Theory, CONCUR 1994*, volume 836 of *LNCS*, pages 481–496. Springer, 1994. doi:10.1007/978-3-540-48654-1.

39 Alexandra Silva, Filippo Bonchi, Marcello Bonsangue, and Jan Rutten. Generalizing determinization from automata to coalgebras. *Log. Methods Comput. Sci*, 9(1:9), 2013.

40 A. Smirnov. Graded Monads and Rings of Polynomials. *J. Math. Sci.*, 151:3032–3051, 2008.

41 Natsuki Urabe and Ichiro Hasuo. Coalgebraic Infinite Traces and Kleisli Simulations. In *Algebra and Coalgebra in Computer Science, CALCO 2015*, volume 35 of *LIPIcs*, pages 320–335. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2015.

42 R. van Glabbeek. The Linear Time – Branching Time Spectrum I; The Semantics of Concrete, Sequential Processes. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 3–99. Elsevier, 2001. doi:10.1016/B978-044482830-9/50019-9.

43 Gerco van Heerdt, Justin Hsu, Joël Ouaknine, and Alexandra Silva. Convex Language Semantics for Nondeterministic Probabilistic Automata. In *Theoretical Aspects of Computing. ICTAC 2018*, volume 11187 of *LNCS*, pages 472–492. Springer, 2018. doi:10.1007/978-3-030-02508-3.

44 James Worrell. On the final sequence of a finitary set functor. *Theor. Comput. Sci.*, 338:184–199, 2005.

Bialgebraic Semantics for String Diagrams

Filippo Bonchi
University of Pisa, Italy

Robin Piedeleu
University College London, UK

Pawel Sobocinski
University of Southampton, UK

Fabio Zanasi
University College London, UK

Abstract

Turi and Plotkin's bialgebraic semantics is an abstract approach to specifying the operational semantics of a system, by means of a distributive law between its syntax (encoded as a monad) and its dynamics (an endofunctor). This setup is instrumental in showing that a semantic specification (a coalgebra) satisfies desirable properties: in particular, that it is compositional.

In this work, we use the bialgebraic approach to derive well-behaved structural operational semantics of *string diagrams*, a graphical syntax that is increasingly used in the study of interacting systems across different disciplines. Our analysis relies on representing the two-dimensional operations underlying string diagrams in various categories as a monad, and their bialgebraic semantics in terms of a distributive law for that monad.

As a proof of concept, we provide bialgebraic compositional semantics for a versatile string diagrammatic language which has been used to model both signal flow graphs (control theory) and Petri nets (concurrency theory). Moreover, our approach reveals a correspondence between two different interpretations of the Frobenius equations on string diagrams and two synchronisation mechanisms for processes, à la Hoare and à la Milner.

2012 ACM Subject Classification Theory of computation → Categorical semantics

Keywords and phrases String Diagram, Structural Operational Semantics, Bialgebraic semantics

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.37

Related Version A full version of the paper is available at [7], `https://arxiv.org/abs/1906.01519`.

Acknowledgements RP and FZ acknowledge support from EPSRC grant EP/R020604/1.

1 Introduction

Starting from the seminal works of Hoare and Milner, there was an explosion [16,17,27,36,42] of interest in process calculi: formal languages for specifying and reasoning about concurrent systems. The beauty of the approach, and often the focus of research, lies in *compositionality*: essentially, the behaviour of composite systems – usually understood as some kind of process equivalence, the most famous of which is bisimilarity – ought to be a function of the behaviour of its components. The central place of compositionality led to the study of syntactic formats for semantic specifications [4,19,25]; succinctly stated, syntactic operations with semantics defined using such formats are homomorphic wrt the semantic space of behaviours.

Another thread of concurrency theory research [26,30,40] uses graphical formalisms, such as Petri nets. These often have the advantage of highlighting connectivity, distribution and the communication topology of systems. They tend to be popular with practitioners in part because of their intuitive and human-readable depictions, an aspect that should not be

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 37; pp. 37:1–37:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

underestimated: indeed, pedagogical texts introducing CCS [27] and CSP [36] often resort to pictures that give intuition about topological aspects of syntactic specifications. However, compositionality has not – historically – been a principal focus of research.

In this paper we propose a framework that allows us to eat our cake and have it too. We use *string diagrams* [43] which have an intuitive graphical rendering, but also come with algebraic operations for composition. String diagrams combine the best of both worlds: they are a (2-dimensional) syntax, but also convey important topological information about the systems they specify. They have been used in recent years to give compositional accounts of quantum circuits [1,18], signal flow graphs [2,10,21], Petri nets [6], and electrical circuits [3,24], amongst several other applications.

Our main contribution is the adaptation of Turi and Plotkin's bialgebraic semantics (*abstract GSOS*) [32,45] for string diagrams. By doing so, we provide a principled justification and theoretical framework for giving definitions of operational semantics to the generators and operations of string diagrams, which are those of monoidal categories. More precisely we deal with string diagrams for symmetric monoidal categories which organise themselves as arrows of a particularly simple and well-behaved class known as *props*. Similar operational definitions have been used in the work on the algebra of Span(Graph) [31], tile logic [23], the wire calculus [44] and recent work on modelling signal flow graphs and Petri nets [6, 10]. In each case, semantics was given either monolithically or via a set of SOS rules. The link with bialgebraic framework – developed in this paper – provides us a powerful theoretical tool that (i) justifies these operational definitions and (ii) guarantees compositionality.

In a nutshell, in the bialgebraic approach, the syntax of a language is the initial algebra (the algebra of terms) T_Σ for a signature functor Σ. A certain kind of distributive law, an *abstract GSOS* specification [45], induces a coalgebra (a state machine) $\beta\colon T_\Sigma \to \mathcal{F}T_\Sigma$ capturing the operational semantics of the language. The final $\mathcal{F}$-coalgebra Ω provides the denotational universe: intuitively, the space of all possible behaviours. The unique coalgebra map $[\![\cdot]\!]_\beta\colon T_\Sigma \to \Omega$ represents the denotational semantics assigning to each term its behaviour.

$$\begin{tikzcd}
T_\Sigma \arrow[r, dashed, "{[\![\cdot]\!]_\beta}"] \arrow[d, "\beta"'] & \Omega \arrow[d] \\
\mathcal{F}(T_\Sigma) \arrow[r, "{\mathcal{F}([\![\cdot]\!]_\beta)}"'] & \mathcal{F}(\Omega)
\end{tikzcd} \tag{1}$$

The crucial observation is that (1) lives in the category of Σ-algebras: Ω also carries a Σ-algebra structure and the denotational semantics is an algebra homomorphism. This means that the behaviour of a composite system is determined by the behaviour of the components, e.g. $[\![s+t]\!] = [\![s]\!] + [\![t]\!]$, for an operation $+$ in Σ.

We show that the above framework can be adapted to the algebra of string diagrams. The end result is a picture analogous to (1), but living in the category of props and prop morphism. As a result, the denotational map is a prop morphism, and thus guarantees compositionality with respect to sequential and parallel composition of string diagrams.

Adapting the bialgebraic approach to the 2-dimensional syntax of props requires some technical work since, e.g. the composition operation of monoidal categories is a dependent operation. For this reason we need to adapt the usual notion of a syntax endofunctor on the category of sets; instead we work in a category Sig whose objects are spans $\mathbb{N} \leftarrow \Sigma \to \mathbb{N}$, with the two legs giving the number of dangling wires on the left and right of each diagram. The operations of props are captured as a Sig-endofunctor, which yields string-diagrams-as-initial-algebra, and a quotient of the resulting free monad, whose algebras are precisely props.

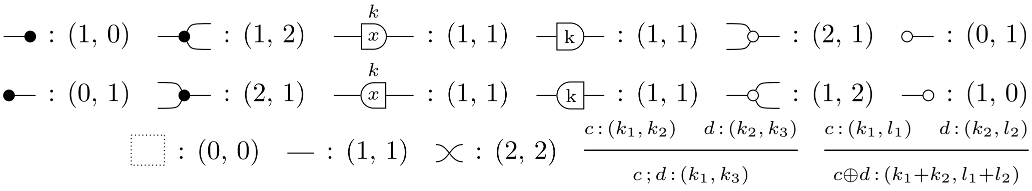


Figure 1 Sorting discipline for $\mathsf{Circ}_{\mathsf{R}}$.

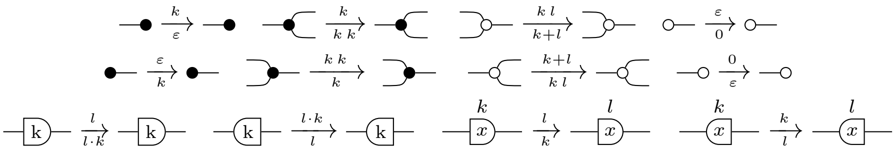


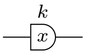
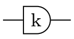
Figure 2 Structural Operational Semantics for the generators of $\mathsf{Circ}_{\mathsf{R}}$. Intuitively, from left to right, these are elementary connectors modelling discard, copy, one-place register, multiplication by a scalar, addition, and the constant zero.

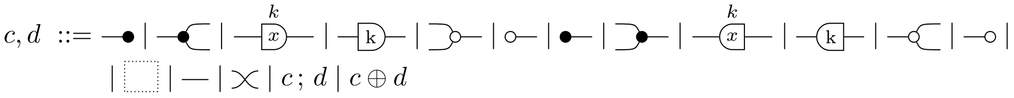
In addition to the basic laws of props, we also consider the further imposition of the equations of special Frobenius algebras. We illustrate the role of this algebraic structure with our running example, a string diagrammatic process calculus $\mathsf{Circ}_{\mathsf{R}}$ that has two Frobenius structures and can be equipped with two different semantics, one which provides a compositional account of signal flow graphs for linear time invariant dynamical systems [10], and one which is a compositional account of Petri nets [6].

We conclude with an observation that ties our work back to classical concepts of process calculi and show that the two Frobenius structures of $\mathsf{Circ}_{\mathsf{R}}$ are closely related to two different, well-known synchronisation patterns, namely those of Hoare's CSP [27] and Milner's CCS [36].

Structure of the paper. In §2 we introduce our main example and recall some preliminaries, followed by a recapitulation of bialgebraic approach in §3. We develop the technical aspects of string-diagrams-as-syntax in §4 and adapt the bialgebraic approach in §5. Finally, we exhibit the connection with classical synchronisation mechanisms in §6 and conclude in §7.

2 Motivating Example

As our motivating example, we recall from [6, 9, 11] a basic language $\mathsf{Circ}_{\mathsf{R}}$ given by the grammar below. Values k in  and  range over elements of a given semiring R.



(2)

(3)

The language does not feature variables; on the other hand, a simple sorting discipline is necessary. A sort is a pair (n, m), with $n, m \in \mathbb{N}$. Henceforth we will consider only terms sortable according to the rules in Figure 1. An easy induction confirms uniqueness of sorting.

The operational meaning of terms is defined recursively by the structural rules in Figs. 2 and 3 where k, l range over R and $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ over $\mathsf{R}^\star$, the set of words over R. We denote the empty word by ε and concatenation of $\boldsymbol{a}, \boldsymbol{b}$ by $\boldsymbol{ab}$. As expected $+, \cdot$ and 0 denote respectively

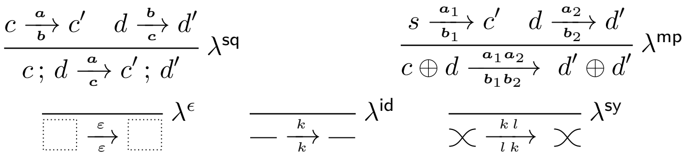
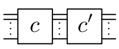
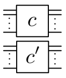
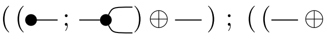
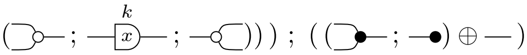


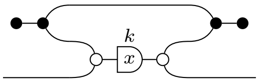
Figure 3 Structural operational semantics for the operations of $\mathsf{Circ}_{\mathsf{R}}$.

the sum, the product and zero of the semiring R. For any term $c\colon (n, m)$, the rules yield a labelled transition system where each transition has form $c \xrightarrow[\boldsymbol{b}]{\boldsymbol{a}} d$. By induction, it is immediate that d has the same sort as c, the word $\boldsymbol{a}$ has length n, and $\boldsymbol{b}$ has length m.

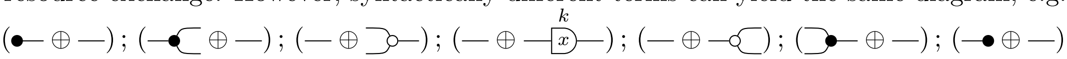
Our chief focus in this paper is the study of semantics specifications of the kind given in Figs. 2 and 3. So far, the technical difference with typical GSOS examples [4] is the presence of a sorting discipline. A more significant difference, which we will now highlight, is that sorted terms are considered up-to the laws of symmetric monoidal categories. As such, they are "2-dimensional syntax" and enjoy a pictorial representation in terms of string diagrams.

2.1 From Terms to String Diagrams

In (2)-(3) we purposefully used a graphical rendering of the components. Indeed, terms of $\mathsf{Circ}_{\mathsf{R}}$ are usually represented graphically, according to the convention that $c\,;\,c'$ is drawn  and $c \oplus c'$ is drawn  . For instance, the term   is depicted as the following diagram.

p_k ::=  (4)

Given this graphical convention, a sort gives the number of dangling wires on each side of the diagram induced by a term. A transition $c \xrightarrow[\boldsymbol{b}]{\boldsymbol{a}} d$ means that c may evolve to d when the values on the dangling wires on the left are $\boldsymbol{a}$ and those on the right are $\boldsymbol{b}$. When R is the natural numbers, the diagram in (4) behaves as a place of a Petri nets containing k tokens: any number of tokens can be inserted from its left and at most k tokens can be removed from its right. Indeed, by the rules in Figs. 2 and 3, $p_k \xrightarrow[o]{i} p_{k'}$ iff $o \leq k$ and $k' = i + k - o$.

The graphical notation is appealing as it highlights connectivity and the capability for resource exchange. However, syntactically different terms can yield the same diagram, e.g.  also yields (4). Indeed, one defines diagrams to be terms modulo *structural congruence*, denoted by $\equiv$. This is the smallest congruence over terms generated by the equations of strict symmetric monoidal categories (SMCs):

$$(f \oplus g) \oplus h \equiv f \oplus (g \oplus h) \qquad (\epsilon \oplus f) \equiv f \qquad (f \oplus \epsilon) \equiv f \qquad \sigma_{1,1}\,;\,\sigma_{1,1} \equiv id_2 \tag{5}$$

$$(f\,;\,g) \oplus (h\,;\,i) \equiv (f \oplus h)\,;\,(g \oplus i) \qquad (f\,;\,g)\,;\,h \equiv f\,;\,(g\,;\,h) \qquad (f\,;\,id_m) \equiv f \tag{6}$$

$$(id_n; f) \equiv f \qquad (\sigma_{1,n}; (f \oplus id_1)) \equiv (id_1 \oplus f); \sigma_{1,m} \qquad (\sigma_{n,1}; (id_1 \oplus g)) \equiv (g \oplus id_1); \sigma_{m,1} \tag{7}$$

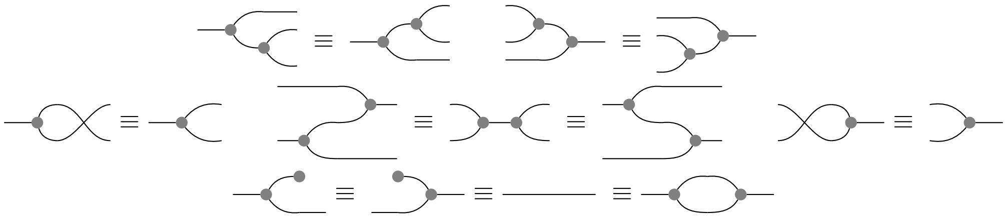


Figure 4 Axioms of special Frobenius bimonoids.

where identities $id_n : (n, n)$ and symmetries $\sigma_{n,m} : (n + m, m + n)$ can be recursively defined starting from $id_0 :=$ □ and $\sigma_{1,1} :=$ ⤫. Therefore, sorted diagrams $c : (n, m)$ are the arrows $n \to m$ of an SMC with objects the natural numbers, also called a prop [35].

2.2 Frobenius Bimonoids

We will also consider additional algebraic structure for the black (—●, —●⊂, ●—, ⊃●—) and the white (○—, ⊃○—, —○, —○⊂) components. When R is the field of reals, Circ_R models linear dynamical systems [2,11,21] and both the black and the white structures form special Frobenius bimonoids, meaning the axioms of Fig. 4 hold, replacing the gray circles by either black or white. When R is the semiring of natural numbers, Circ_R models Petri nets [6] and only the black structure satisfies these equations. In § 6, we shall see that the black Frobenius structure gives rise to the synchronisation mechanism used by Hoare in CSP [28], while the white Frobenius structure to that used by Milner in CCS [36].

3 Background: Bialgebras and GSOS Specifications

For more detailed background and simple examples showcasing the notions recalled below, we refer the reader to the extended version of the present work [7].

Distributive laws and bialgebras. A *distributive law* of a monad $(\mathcal{T}, \eta, \mu)$ over an endofunctor $\mathcal{F}$ is a natural transformation $\lambda\colon \mathcal{T}\mathcal{F} \Rightarrow \mathcal{F}\mathcal{T}$ s.t. $\lambda \circ \eta_\mathcal{F} = \mathcal{F}\eta$ and $\lambda \circ \mu_\mathcal{F} = \mathcal{F}\mu \circ \lambda_\mathcal{T} \circ \mathcal{T}\lambda$. A λ-*bialgebra* is a triple (X, α, β) s.t. (X, α) is an Eilenberg-Moore algebra for $\mathcal{T}$, (X, β) is a $\mathcal{F}$-coalgebra and $\mathcal{F}\alpha \circ \lambda_X \circ \mathcal{T}\beta = \beta \circ \alpha$. Bialgebra morphisms are both $\mathcal{T}$-algebra and $\mathcal{F}$-coalgebra morphisms.

Given a coalgebra $\beta\colon X \to \mathcal{F}\mathcal{T}X$ for a monad $(\mathcal{T}, \eta, \mu)$ and a functor $\mathcal{F}$, if there exists a distributive law $\lambda\colon \mathcal{T}\mathcal{F} \Rightarrow \mathcal{F}\mathcal{T}$, one can form a coalgebra $\beta^\sharp\colon \mathcal{T}X \to \mathcal{F}\mathcal{T}X$ defined as $\mathcal{T}X \xrightarrow{\mathcal{T}\beta} \mathcal{T}\mathcal{F}\mathcal{T}X \xrightarrow{\lambda_{\mathcal{T}X}} \mathcal{F}\mathcal{T}\mathcal{T}X \xrightarrow{\mathcal{F}\mu} \mathcal{F}\mathcal{T}X$. Most importantly, $(\mathcal{T}X, \mu, \beta^\sharp)$ is a λ-bialgebra.

Free monads. We recall the construction of the monad $\mathcal{F}^\dagger\colon \mathcal{C} \to \mathcal{C}$ *freely generated* by a functor $\mathcal{F}\colon \mathcal{C} \to \mathcal{C}$. Assume that $\mathcal{C}$ has coproducts and that, for all objects X of $\mathcal{C}$, there exists an initial $X + \mathcal{F}$-algebra that we denote as $X + \mathcal{F}(\mathcal{F}^\dagger X) \xrightarrow{[\eta_X,\kappa_X]} \mathcal{F}^\dagger X$. It is easy to check that the assignment $X \mapsto \mathcal{F}^\dagger X$ induces a functor $\mathcal{F}^\dagger\colon \mathcal{C} \to \mathcal{C}$. The map $\eta_X\colon X \to \mathcal{F}^\dagger X$ gives rise to the unit of the monad; the multiplication $\mu_X\colon \mathcal{F}^\dagger\mathcal{F}^\dagger X \to \mathcal{F}^\dagger X$ is the unique algebra morphism from the initial $\mathcal{F}^\dagger X + \mathcal{F}$-algebra to the algebra $\mathcal{F}^\dagger X + \mathcal{F}(\mathcal{F}^\dagger X) \xrightarrow{[id,\kappa_X]} \mathcal{F}^\dagger X$.

GSOS specifications. An *abstract GSOS specification* is a natural transformation $\lambda\colon \mathcal{S}\mathcal{F} \Rightarrow \mathcal{F}\mathcal{S}^\dagger$, where $\mathcal{F}$ is a functor representing the coalgebraic behaviour, $\mathcal{S}$ is a functor representing the syntax. It is important to recall the following fact.

▶ **Proposition 1** ([34])**.** *Any GSOS spec.* $\lambda\colon \mathcal{S}\mathcal{F} \Rightarrow \mathcal{F}\mathcal{S}^\dagger$ *yields a distrib. law* $\lambda^\dagger\colon \mathcal{S}^\dagger\mathcal{F} \Rightarrow \mathcal{F}\mathcal{S}^\dagger$*.*

Coproduct of GSOS specifications. Suppose we have two functors $\mathcal{S}_1, \mathcal{S}_2\colon \mathcal{C} \to \mathcal{C}$ capturing two syntaxes, a functor $\mathcal{F}\colon \mathcal{C} \to \mathcal{C}$ for the coalgebraic behaviour, and two GSOS specifications $\lambda_1\colon \mathcal{S}_1\mathcal{F} \Rightarrow \mathcal{F}\mathcal{S}_1^\dagger$ and $\lambda_2\colon \mathcal{S}_2\mathcal{F} \Rightarrow \mathcal{F}\mathcal{S}_2^\dagger$. Then we can construct a GSOS specification $\lambda_1 \cdot \lambda_2\colon (\mathcal{S}_1 + \mathcal{S}_2)\mathcal{F} \Rightarrow \mathcal{F}(\mathcal{S}_1 + \mathcal{S}_2)^\dagger$. The details are in [7].

Quotients of monads and distributive laws. Given the correspondence between finitary monads and algebraic theories [29], it natural to consider *quotients* of monads by additional equations. Following [13, 15, 41], for a monad $\mathcal{T}$ on a category $\mathcal{C}$, $\mathcal{T}$*-equations* can be defined as a tuple $\mathbb{E} = (\mathcal{A}, l, r)$ consisting of a functor $\mathcal{A}\colon \mathcal{C} \to \mathcal{C}$ and natural transformations $l, r\colon \mathcal{A} \Rightarrow \mathcal{T}$. The intuition is that $\mathcal{A}$ acts as the variables of each equation, whose left- and right-hand sides are l and r, respectively. Assuming mild conditions that generalise the properties of Set (see [41, Ass. 7.1.2]), one constructs the *quotient* of $\mathcal{T}$ by $\mathcal{T}$-equations. The conditions hold in our setting: categories of presheaves over a discrete index category.

▶ **Proposition 2** (cf. [41])**.** *If* $\mathcal{C} = \mathsf{Set}^\mathcal{D}$ *for discrete* $\mathcal{D}$*,* $\mathcal{T}$*-equations* $\mathbb{E}$ *yield a monad* $\mathcal{T}_{/\mathbb{E}}\colon \mathcal{C} \to \mathcal{C}$ *with algebras precisely* $\mathcal{T}$*-algebras* $\mathcal{T}A \xrightarrow{\alpha} A$ *that satisfy* $\mathbb{E}$*, in the sense that* $\alpha \circ l_A = \alpha \circ r_A$*. Moreover, there exists a monad morphism* $q^\mathbb{E}\colon \mathcal{T} \to \mathcal{T}_{/\mathbb{E}}$ *with epi components.*

One may also quotient distributive laws, provided these are compatible with the new equations. Fix an endofunctor $\mathcal{F}$ and a monad $\mathcal{T}$ on $\mathsf{Set}^\mathcal{D}$, together with $\mathcal{T}$-equations $\mathbb{E} = (\mathcal{A}, l, r)$. We say that a distributive law $\lambda\colon \mathcal{T}\mathcal{F} \Rightarrow \mathcal{F}\mathcal{T}$ *preserves equations* $\mathbb{E}$ if, for all $A \in \mathcal{C}$, the following diagram commutes:

$$\begin{tikzcd} \mathcal{A}\mathcal{F}A \arrow[r, shift left, "l_{\mathcal{F}A}"] \arrow[r, shift right, "r_{\mathcal{F}A}"'] & \mathcal{T}\mathcal{F}A \arrow[r, "\lambda_A"] & \mathcal{F}\mathcal{T}A \arrow[r, "\mathcal{F}q^\mathbb{E}_A"] & \mathcal{F}\mathcal{T}_{/\mathbb{E}}A \end{tikzcd} .$$

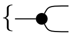
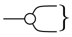
▶ **Proposition 3** (cf. [41])**.** *If* $\lambda\colon \mathcal{T}\mathcal{F} \to \mathcal{F}\mathcal{T}$ *preserves equations* $\mathbb{E}$ *then there exists a (unique) distributive law* $\lambda_{/\mathbb{E}}\colon \mathcal{T}_{/\mathbb{E}}\mathcal{F} \Rightarrow \mathcal{F}\mathcal{T}_{/\mathbb{E}}$ *such that* $\lambda_{/\mathbb{E}} \circ q^\mathbb{E}\mathcal{F} = \mathcal{F}q^\mathbb{E} \circ \lambda$*.*

4 Diagrammatic Syntax as Monads

4.1 The Category of Signatures

Syntax and semantics of string diagrams will be specified in the category $\mathsf{Sig} := \mathsf{Span}(\mathsf{Set})(\mathbb{N}, \mathbb{N})$, where objects are spans $\mathbb{N} \leftarrow \Sigma \rightarrow \mathbb{N}$ in Set and arrows are span morphisms: given objects $\mathbb{N} \xleftarrow{s} X \xrightarrow{t} \mathbb{N}$ and $\mathbb{N} \xleftarrow{s'} \Sigma' \xrightarrow{t'} \mathbb{N}$, an arrow is a function $f\colon \Sigma \to \Sigma'$ such that $t' \circ f = t$ and $s' \circ f = s$. We think of an object of Sig as a *signature*, i.e. a set of symbols Σ equipped with arity and coarity functions $a, c\colon \Sigma \to \mathbb{N}$. We write $\Sigma(n, m)$ for the set $\{d \in \Sigma \mid \langle a, c\rangle(d) = (n, m)\}$ of operations with arity n and coarity m. Note that we allow coarities different from 1: this is because string diagrams express *monoidal* algebraic theories, not merely *cartesian* ones.

Since the objects in Sig are spans with identical domain and codomain, we will often write Σ for the entire span $\mathbb{N} \xleftarrow{a} \Sigma \xrightarrow{c} \mathbb{N}$. In particular, $\mathbb{N}$ means the identity span $\mathbb{N} \xleftarrow{id} \mathbb{N} \xrightarrow{id} \mathbb{N}$.

▶ **Example 4.** Recall the language Circ_R from § 2. Line (2) of its syntax together with the first two lines of the sorting discipline in Fig. 1 define a signature Σ: every axiom $d : (n, m)$ gives the symbol d arity n and coarity m. For instance, $\Sigma(1, 2) = \{$, $\}$.

For computing (co)limits, it is useful to observe that Sig is isomorphic to the presheaf category $\mathsf{Set}^{\mathbb{N}\times\mathbb{N}}$, where $\mathbb{N}\times\mathbb{N}$ is the discrete category with objects pairs $(n,m)\in\mathbb{N}\times\mathbb{N}$.

4.2 Functors on Signatures

We turn to (co)algebras of endofunctors $\mathcal{F}\colon \mathsf{Sig}\to\mathsf{Sig}$ generated by the following grammar:

$$\mathcal{F} \quad ::= \quad Id \mid \Sigma \mid \mathbb{N} \mid \mathcal{F}\,;\mathcal{F} \mid \mathcal{F}\oplus\mathcal{F} \mid \mathcal{F}+\mathcal{F} \mid \mathcal{F}\times\mathcal{F} \mid \overline{\mathcal{G}}$$

where $\mathcal{G}$ ranges over functors $\mathcal{G} : \mathsf{Set}\to\mathsf{Set}$ and Σ is a span $\mathbb{N}\leftarrow\Sigma\to\mathbb{N}$. In more detail:

- $Id\colon \mathsf{Sig}\to\mathsf{Sig}$ is the identity functor.
- $\Sigma\colon \mathsf{Sig}\to\mathsf{Sig}$ is the constant functor mapping every object to $\mathbb{N}\leftarrow\Sigma\to\mathbb{N}$ and every arrow to id_Σ; an important special case is $\mathbb{N}\colon \mathsf{Sig}\to\mathsf{Sig}$ the constant functor to $\mathbb{N}\xleftarrow{id}\mathbb{N}\xrightarrow{id}\mathbb{N}$.
- $(\cdot)\,;(\cdot)\colon \mathsf{Sig}^2\to\mathsf{Sig}$ is *sequential composition* for signatures. On objects, $\Sigma_1\,;\Sigma_2$ is

  $$\mathbb{N}\xleftarrow{s_1\circ\pi_1}\{(d_1,d_2)\in\Sigma_1\times\Sigma_2 \mid t_1(d_1)=s_2(d_2)\}\xrightarrow{t_2\circ\pi_2}\mathbb{N}.$$

  Since the above is a Set-pullback, the action on arrows is inducted by the universal property. Note that, up to iso, $(\cdot)\,;(\cdot)\colon \mathsf{Sig}^2\to\mathsf{Sig}$ is associative with unit $\mathbb{N}\colon\mathsf{Sig}\to\mathsf{Sig}$.
- $(\cdot)\oplus(\cdot)\colon \mathsf{Sig}^2\to\mathsf{Sig}$ is *parallel composition* for signatures, with $\Sigma_1\oplus\Sigma_2$ given by:

  $$\mathbb{N}\xleftarrow{+\circ(s_1\times s_2)}\Sigma_1\times\Sigma_2\xrightarrow{+\circ(t_1\times t_2)}\mathbb{N}$$

  where $+\colon \mathbb{N}\times\mathbb{N}\to\mathbb{N}$ is usual $\mathbb{N}$-addition. Again $(\cdot)\oplus(\cdot)\colon\mathsf{Sig}^2\to\mathsf{Sig}$ associates up to iso.
- For the remaining functors, we use the fact that $\mathsf{Sig}\cong\mathsf{Set}^{\mathbb{N}\times\mathbb{N}}$, which guarantees (co)completeness, with limits and colimits constructed pointwise in Set. Thus, for spans Σ_1 and Σ_2, their coproduct is $\mathbb{N}\xleftarrow{[s_1,s_2]}\Sigma_1+\Sigma_2\xrightarrow{[t_1,t_2]}\mathbb{N}$ and the carrier of the product is $\{(d_1,d_2)\mid s_1(d_1)=s_2(d_2) \text{ and } t_1(d_1)=t_2(d_2)\}$, with the two obvious morphisms to $\mathbb{N}$.
- The isomorphism $\mathsf{Sig}\cong\mathsf{Set}^{\mathbb{N}\times\mathbb{N}}$ also yields the extension of an arbitrary endofunctor $\mathcal{G}\colon\mathsf{Set}\to\mathsf{Set}$ to a functor $\bar{\mathcal{G}}\colon\mathsf{Sig}\to\mathsf{Sig}$ defined by post-composition with $\mathcal{G}$, that is $\bar{\mathcal{G}}(\Sigma)=\mathcal{G}\circ\Sigma$ for all $\Sigma\colon\mathbb{N}\times\mathbb{N}\to\mathsf{Set}$. In particular, we shall often use the functor $\overline{\mathcal{P}_\kappa}$ obtained by post-composition with the κ-bounded powerset functor $\mathcal{P}_\kappa\colon\mathsf{Set}\to\mathsf{Set}$.[1]

Next we use these endofunctors to construct monads that capture the two-dimensional algebraic structure of string diagrams. In § 4.3 we construct the monad encoding the symmetric monoidal structure of props and in § 4.4 we construct the monad for the Frobenius structure of Carboni-Walters props. Later, in § 5, we shall use these monads to define compositional bialgebraic semantics for string diagrams of each of these categorical structures.

4.3 The Prop Monad

Here we define a monad on Sig with algebras precisely props: symmetric strict monoidal categories with objects the natural numbers, where the monoidal product on objects is addition. Together with identity-on-objects symmetric monoidal functors they form a category **PROP**. The first step is to encapsulate the operations of props as a Sig-endofunctor.

$$\mathcal{S}_{\mathsf{SM}} := (Id\,;Id)+\iota+(Id\oplus Id)+\epsilon+\sigma\colon\mathsf{Sig}\to\mathsf{Sig}. \tag{8}$$

[1] Boundedness is needed to ensure the existence of a final coalgebra, see § 5.1. In our leading example $\mathsf{Circ}_{\mathsf{R}}$, κ can be taken to be the cardinality of the semiring R.

In the type of $\mathcal{S}_{\mathsf{SM}}$, $Id\,;\,Id\colon \mathsf{Sig} \to \mathsf{Sig}$ is sequential composition and ι the identity arrow on object 1, i.e. the constant functor to $\mathbb{N} \xleftarrow{h} \{id_1\} \xrightarrow{h} \mathbb{N}$, with $h\colon id_1 \mapsto 1$. Similarly, $Id \oplus Id$ is the monoidal product with unit ϵ, i.e. the constant functor to $\mathbb{N} \xleftarrow{q} \{0\} \xrightarrow{q} \mathbb{N}$, with $q\colon 0 \mapsto 0$. Finally, σ is the basic symmetry: the constant functor to $\mathbb{N} \xleftarrow{f} \{\sigma_{1,1}\} \xrightarrow{f} \mathbb{N}$, with $f\colon \sigma_{1,1} \mapsto 2$.

The free monad $\mathcal{S}^{\dagger}_{\mathsf{SM}}$ on $\mathcal{S}_{\mathsf{SM}}$ is the functor mapping a span Σ to the span of Σ-terms obtained by sequential and parallel composition, together with symmetries and identities – with the identity id_n defined by parallel composition of n copies of id_1.

Algebras for this monad are spans Σ together with span morphisms $identity\colon \iota \to \Sigma$, $composition\colon \Sigma\,;\,\Sigma \to \Sigma$, $parallel\colon \Sigma \oplus \Sigma \to \Sigma$, $unit\colon \epsilon \to \Sigma$, and $swap\colon \sigma \to \Sigma$. This information *almost* defines a prop $\mathcal{C}_{\Sigma}$: the carrier Σ of the span is the set of arrows of $\mathcal{C}_{\Sigma}$, containing special arrows id_n and $\sigma_{n,m}$ for identities and symmetries, *compose* assigns to every pairs of composable arrows their composition, and $\oplus$ assigns to every pair of arrows their monoidal product. The missing data is the usual equations (5)-(7) of symmetric monoidal categories. Thus, in order to obtain props as algebras, we quotient the monad $\mathcal{S}^{\dagger}_{\mathsf{SM}}$ by those equation, expressed abstractly as a triple $\mathbb{E}_{\mathsf{SM}} = (\mathcal{A}, l, r)$, as described in § 3. The functor $\mathcal{A}\colon \mathsf{Sig} \to \mathsf{Sig}$, defined below, has summands following the order (5)-(7):

$$(Id \oplus Id \oplus Id) + Id + Id + \sigma + ((Id\,;Id) \oplus (Id\,;Id)) + (Id\,;Id\,;Id) + Id + Id + Id^{+1} + Id^{+1} \quad (9)$$

Here, Id^{+1} is the functor adding 1 to the arity/coarity of each element of a given span $\mathbb{N} \xleftarrow{a} \Sigma \xrightarrow{c} \mathbb{N}$. We also need natural transformations $l, r\colon \mathcal{A} \to \mathcal{S}^{\dagger}_{\mathsf{SM}}$ that define the left- and right-hand side of each equation. For instance, for fixed $\Sigma \in \mathsf{Sig}$ and $(n, m) \in \mathbb{N} \times \mathbb{N}$:

- an element of $\Sigma\,;\,\Sigma\,;\,\Sigma$ (sixth summand of (9)) is a tuple (f, g, h) of Σ-elements, where f is of type (n, w), g of type (w, v), and h of type (v, m), for arbitrary $w, v \in \mathbb{N}$. We let l_{Σ} map (f, g, h) to the term $(f\,;\,g)\,;\,h$ of type (n, m) in $\mathcal{S}^{\dagger}_{\mathsf{SM}}(\Sigma)$, and r_{Σ} to the term $f\,;\,(g\,;\,h)$. Thus this component gives the second equation in (6) (associativity).
- the seventh summand Id in (9) yields a Σ-term f, which $l_{\Sigma}\colon \Sigma \to \mathcal{S}^{\dagger}_{\mathsf{SM}}(\Sigma)$ maps to $f\,;\,id_m$ and $r_{\sigma}\colon \Sigma \to \mathcal{S}^{\dagger}_{\mathsf{SM}}(\Sigma)$ maps to f, thus yielding the final equation in (6).
- an element in Σ^{+1} (last summand of (9)) of type $(n+1, m+1)$ is a Σ-term g of type (n, m), which is mapped by l_{Σ} to $(\sigma_{n,1}\,;\,(id_1 \oplus g))$ and by r_{σ} to $(g \oplus id_1)\,;\,\sigma_{m,1}$, both elements of $\mathcal{S}^{\dagger}_{\mathsf{SM}}(\Sigma)$ of type $(n+1, m+1)$, thus giving the final equation in (7).

The remainder of the definition of $l, r\colon \mathcal{A} \to \mathcal{S}^{\dagger}_{\mathsf{SM}}$, handles the remaining equations in (5)-(7), and should be clear from the above. Now, using Proposition 2, we quotient the monad $\mathcal{S}^{\dagger}_{\mathsf{SM}}$ by $(\mathcal{A}, l, r)$, obtaining a monad that we call $\mathcal{S}_{\mathsf{PROP}}$. We can then conclude by construction that the Eilenberg-Moore category $EM(\mathcal{S}_{\mathsf{PROP}})$ for the monad $\mathcal{S}_{\mathsf{PROP}}$ (with objects the $\mathcal{S}_{\mathsf{PROP}}$-algebras, and arrows the $\mathcal{S}_{\mathsf{PROP}}$-algebra homomorphisms) is precisely **PROP**.

▶ **Proposition 5.** *$EM(\mathcal{S}_{PROP}) \cong$* **PROP**.

▶ **Example 6.** The monad $\mathcal{S}_{\mathsf{PROP}}$ takes Σ to the prop freely generated by Σ. Taking Σ as in Example 4, one obtains $\mathcal{S}_{\mathsf{PROP}}(\Sigma)$ with arrows $n \to m$ string diagrams of $\mathsf{Circ}_{\mathsf{R}}$ of sort (n, m).

4.4 The Carboni-Walters Monad

The treatment we gave to props may be applied to other categorical structures. For space reasons, we only consider one additional such structure: *Carboni-Walters* (CW) props, also called "hypergraph categories" [22]. Here each object n carries a distinguished special Frobenius bimonoid compatible with the monoidal product: it can be defined recursively using parallel compositions of the Frobenius structure on the generating object 1.

▶ **Definition 7.** *A* CW prop *is a prop with morphisms* —●< $: 1 \to 2$, —● $: 1 \to 0$, >●— $: 2 \to 1$, ●— $: 0 \to 1$ *satisfying the equations of special Frobenius bimonoids (Fig. 4).*

CW props with prop morphisms preserving the Frobenius bimonoid form a subcategory **CW** of **PROP**. We can now extend the prop monad of § 4.3 to obtain a monad with algebras CW props. The signature is that of a prop with the additional Frobenius structure. Let —●< : $\mathsf{Sig} \to \mathsf{Sig}$ be the functor constant at $\mathbb{N} \xleftarrow{s} \{$—●<$\} \xrightarrow{t} \mathbb{N}$ with $s($—●<$) = 1$ and $t($—●<$) = 2$. Similarly, we introduce the constant functors —● $: \mathsf{Sig} \to \mathsf{Sig}$, >●— $: \mathsf{Sig} \to \mathsf{Sig}$ and ●— $: \mathsf{Sig} \to \mathsf{Sig}$ for the other generators. Let $\mathcal{S}_{\mathtt{FR}} := \mathcal{S}_{\mathtt{PROP}} +$ —●< $+$ —● $+$ >●— $+$ ●—.

We now need to quotient $\mathcal{S}_{\mathtt{FR}}$ by the defining equations of special Frobenius bimonoids (Fig. 4). We omit the detailed encoding of these equations as a triple $\mathbb{E}_{\mathsf{CW}} = (\mathcal{A}_{\mathsf{CW}}, l_{\mathsf{CW}}, r_{\mathsf{CW}})$ since it presents no conceptual difficulty. Let $\mathcal{S}_{\mathtt{CW}}$ be the quotient of $\mathcal{S}_{\mathtt{FR}}$ by these equations. As for props, we obtain $EM(\mathcal{S}_{\mathtt{CW}}) \cong \mathbf{CW}$ by construction.

5 Bialgebraic Semantics for String Diagrams

Now that we have established monads for our categorical structures of interest, we study coalgebras that capture behaviour for string diagrams in these categories, and distributive laws that yield the desired bialgebraic semantics. We fix our "behaviour" functor to

$$\mathcal{F} := \overline{\mathcal{P}_\kappa}(L\,;\,Id\,;\,L)\colon \mathsf{Sig} \to \mathsf{Sig}$$

where $L\colon \mathsf{Sig} \to \mathsf{Sig}$ is the *label functor* constant at the span $\mathbb{N} \xleftarrow{|\cdot|} A^* \xrightarrow{|\cdot|} \mathbb{N}$, with A^* the set of words on some set of labels A. The map $|\cdot|\colon A^* \to \mathbb{N}$ takes $w \in A^*$ to its length $|w| \in \mathbb{N}$. An $\mathcal{F}$-coalgebra is a span morphism $\Sigma \to \overline{\mathcal{P}_\kappa}(L\,;\,\Sigma\,;\,L)$; a function that takes $f \in \Sigma(n,m)$ to a set of transitions (v,g,w) with the appropriate sorts, i.e. $g \in \Sigma(n,m)$, $|v| = n$ and $|w| = m$.

The data of an $\mathcal{F}$-coalgebra $\beta\colon \Sigma \to \overline{\mathcal{P}_\kappa}(L\,;\,\Sigma\,;\,L)$ is that of a transition relation. For instance, fix labels $A = \{a,b\}$ and let $x, y \in \Sigma(1,2)$ and $z \in \Sigma(1,1)$; suppose also that β maps x to $\{(b\,;\,y\,;\,ab), (a\,;\,x\,;\,aa)\}$, y to $\emptyset$ and z to $\{(b\,;\,z\,;\,a)\}$. Then β can be written:

$$x \xrightarrow[ab]{b} y \qquad x \xrightarrow[aa]{a} x \qquad z \xrightarrow[a]{b} z \tag{10}$$

▶ **Example 8.** In our main example, Fig. 2 defines a coalgebra $\beta\colon \Sigma \to \overline{\mathcal{P}_\kappa}(L\,;\,\Sigma\,;\,L)$ where Σ is the signature from Example 4 and the set of labels is R. For instance $\beta($—●$) = \{(k,$ —●$, \varepsilon) \mid k \in \mathsf{R}\}$. Note the κ bounding $\mathcal{P}_\kappa$ is the cardinality of R.

In the sequel we shall construct distributive laws between the above behaviour functor and monads encoding the various categorical structures defined in the previous section.

5.1 Bialgebraic Semantics for Props

The modularity of $\mathcal{S}_{\mathtt{PROP}}$ can be exploited to define a distributive law of the $\mathcal{S}_{\mathtt{PROP}}$ over $\mathcal{F}$. Recall from § 4.3 that $\mathcal{S}_{\mathtt{PROP}}$ is a quotient of $\mathcal{S}_{\mathtt{SM}}^{\dagger}$. We start by letting $\mathcal{F} = \overline{\mathcal{P}_\kappa}(L\,;\,Id\,;\,L)$ interact with the individual summands of $\mathcal{S}_{\mathtt{SM}}$ (see (8)), corresponding to the operations of props. This amounts to defining GSOS specifications:

$$\begin{aligned}
&\lambda^{\mathsf{sq}}\colon \overline{\mathcal{P}_\kappa}(L\,;\,Id\,;\,L)\,;\,\overline{\mathcal{P}_\kappa}(L\,;\,Id\,;\,L) \Rightarrow \overline{\mathcal{P}_\kappa}(L\,;\,(Id\,;\,Id)^{\dagger}\,;\,L) && \text{(sequential composition)}\\
&\lambda^{\mathsf{id}}\colon \iota \Rightarrow \overline{\mathcal{P}_\kappa}(L\,;\,\iota^{\dagger}\,;\,L) && \text{(identity)}\\
&\lambda^{\mathsf{mp}}\colon \overline{\mathcal{P}_\kappa}(L\,;\,Id\,;\,L) \oplus \overline{\mathcal{P}_\kappa}(L\,;\,Id\,;\,L) \Rightarrow \overline{\mathcal{P}_\kappa}(L\,;\,(Id \oplus Id)^{\dagger}\,;\,L) && \text{(monoidal product)}\\
&\lambda^{\epsilon}\colon \epsilon \Rightarrow \overline{\mathcal{P}_\kappa}(L\,;\,\epsilon^{\dagger}\,;\,L) && \text{(product unit)}\\
&\lambda^{\mathsf{sy}}\colon \sigma \Rightarrow \overline{\mathcal{P}_\kappa}(L\,;\,\sigma^{\dagger}\,;\,L) && \text{(symmetry)}
\end{aligned}$$

Definitions of these maps are succinctly given via derivation rules, see Fig. 3.

We explain this in detail for λ^{sq}, the others are similar. Given $\Sigma \in \mathsf{Sig}$, an element of type (n, m) in the domain $\overline{\mathcal{P}_\kappa}(L\,;\,\Sigma\,;\,L)\,;\,\overline{\mathcal{P}_\kappa}(L\,;\,\Sigma\,;\,L)$ is a pair (A, B), where, for some $z \in \mathbb{N}$,

A is a set of triples $(\boldsymbol{a}, c', \boldsymbol{b}) \in L(n,n) \times \Sigma(n,z) \times L(z,z),$ and

B is a set of triples $(\boldsymbol{b}, d', \boldsymbol{c}) \in L(z,z) \times \Sigma(z,m) \times L(m,m).$

Then $\lambda^{\mathsf{sq}}_\Sigma(A,B) := \{(\boldsymbol{a}, c'\,;\,d', \boldsymbol{c}) \mid (\boldsymbol{a}, c', \boldsymbol{b}) \in A,\ (\boldsymbol{b}, d', \boldsymbol{c}) \in B\}$. Following the convention (10), we can write this data as: $\boxed{\xrightarrow[c]{a} c'\,;\,d'} \in \lambda^{\mathsf{sq}}_\Sigma(A,B)$ if $\boxed{\xrightarrow[b]{a} c'} \in A$ and $\boxed{\xrightarrow[c]{b} d'} \in B$. This leads us to the more compact version of λ^{sq} as the transition rule in Fig.3.

Next, take the coproduct of GSOS specifications λ^{sq}, λ^{id}, λ^{mp}, λ^{ϵ} and λ^{sy} (see [7] for the details) to obtain $\lambda\colon \mathcal{S}_{\mathsf{SM}}\mathcal{F} \Rightarrow \mathcal{F}\mathcal{S}^{\dagger}_{\mathsf{SM}}$. By Proposition 1, this yields distributive law $\lambda^{\dagger}\colon \mathcal{S}^{\dagger}_{\mathsf{SM}}\mathcal{F} \Rightarrow \mathcal{F}\mathcal{S}^{\dagger}_{\mathsf{SM}}$.

The last step is to upgrade $\lambda^{\dagger}$ to a distributive law $\lambda^{\dagger}{}_{/\mathsf{SMC}}$ over the quotient $\mathcal{S}_{\mathtt{PROP}}$ of $\mathcal{S}^{\dagger}_{\mathsf{SM}}$ by the equations (5)-(7) of SMCs. By Proposition 3, this is well-defined if $\lambda^{\dagger}$ preserves $\mathbb{E}_{\mathsf{SM}}$. We show compatibility with associativity of sequential composition – the other equations can be verified similarly. This amounts to checking that if $\lambda^{\dagger}$ allows the derivation for $s_1\,;\,(s_2\,;\,s_3)$ as below left, then there exists a derivation for $(s_1\,;\,s_2)\,;\,s_3$ as on the right, and vice-versa.

$$\frac{s_1 \xrightarrow[v]{u} s_1 \qquad \dfrac{s_2 \xrightarrow[w]{v} s_2 \quad s_3 \xrightarrow[x]{w} s_3}{s_2\,;\,s_3 \xrightarrow[x]{v} s_2\,;\,s_3}}{s_1\,;\,(s_2\,;\,s_3) \xrightarrow[x]{u} s_1\,;\,(s_2\,;\,s_3)} \qquad \frac{\dfrac{s_1 \xrightarrow[v]{u} s_1 \quad s_2 \xrightarrow[w]{v} s_2}{s_1\,;\,s_2 \xrightarrow[w]{u} s_1\,;\,s_2} \qquad s_3 \xrightarrow[x]{w} s_3}{(s_1\,;\,s_2)\,;\,s_3 \xrightarrow[x]{u} (s_1\,;\,s_2)\,;\,s_3}$$

By Proposition 3, we can therefore upgrade $\lambda^{\dagger}$ to a distributive law $\lambda^{\dagger}{}_{/\mathsf{SM}}\colon \mathcal{S}_{\mathtt{PROP}}\mathcal{F} \Rightarrow \mathcal{F}\mathcal{S}_{\mathtt{PROP}}$.

We are now ready to construct the compositional semantics as a morphism into the final coalgebra. One starts with a coalgebra $\beta\colon \Sigma \to \mathcal{F}(\mathcal{S}_{\mathtt{PROP}}(\Sigma))$ that describes the behaviour of Σ-operations, assigning to each a set of transitions, as in (10). The difference with (10) is that, because $\mathcal{F}$ is applied to $\mathcal{S}_{\mathtt{PROP}}(\Sigma)$ instead of just Σ, the right-hand side of each transition contains not just a Σ-operation, but a *string diagram*: a Σ-term modulo the laws of SMCs.

As recalled in § 3, using the distributive law $\lambda^{\dagger}{}_{/\mathsf{SM}}$ we can lift $\beta\colon \Sigma \to \mathcal{F}(\mathcal{S}_{\mathtt{PROP}}(\Sigma))$ to a $\lambda^{\dagger}{}_{/\mathsf{SM}}$-bialgebra, $\beta^{\sharp} : \mathcal{S}_{\mathtt{PROP}}(\Sigma) \to \mathcal{F}(\mathcal{S}_{\mathtt{PROP}}(\Sigma))$. Since this is a $\mathcal{F}$-coalgebra, the final $\mathcal{F}$-coalgebra Ω (the existence of which is shown in [7]) yields a semantics $[\![\cdot]\!]_\beta$ as below. The operational semantics of a string diagram c is $\beta^{\sharp}(c)$, obtained from *(i)* transitions for Σ-operations given by β and *(ii)* the derivation rules (Fig. 3) of $\lambda^{\dagger}{}_{/\mathsf{SM}}$. Instead, $[\![c]\!]_\beta$ is the observable behaviour: intuitively, its transition systems modulo bisimilarity.

$$\begin{tikzcd}
\mathcal{S}_{\mathtt{PROP}}(\Sigma) \arrow[r, dashed, "{[\![\cdot]\!]_\beta}"] \arrow[d, "\beta^{\sharp}"] & \Omega \arrow[d] \\
\mathcal{F}(\mathcal{S}_{\mathtt{PROP}}(\Sigma)) \arrow[r, "{\mathcal{F}([\![\cdot]\!]_\beta)}"'] & \mathcal{F}(\Omega)
\end{tikzcd}$$

The bialgebraic semantics framework ensures that $\mathcal{S}_{\mathtt{PROP}}(\Sigma)$ and Ω are $\mathcal{S}_{\mathtt{PROP}}$-algebras, which by Proposition 5 are props. This means that the final coalgebra Ω is a prop and that $[\![\cdot]\!]_\beta$ is a prop morphism, preserving identities, symmetries and guaranteeing compositionality: $[\![s\,;\,t]\!]_\beta = [\![s]\!]_\beta\,;\,[\![t]\!]_\beta$ and $[\![s \oplus t]\!]_\beta = [\![s]\!]_\beta \oplus [\![t]\!]_\beta$.

▶ **Example 9.** Coming back to our running example, in Example 8 we showed that rules in Fig. 2 induce a coalgebra of type $\Sigma \to \mathcal{F}(\Sigma)$. Since each operation in Σ is itself a string diagram (formally, via the unit $\eta_\Sigma\colon \Sigma \to \mathcal{S}_{\mathtt{PROP}}(\Sigma)$), the same rules induce a coalgebra

$\beta : \Sigma \to \mathcal{F}\mathcal{S}_{\mathtt{PROP}}(\Sigma)$, which has the type required for the above construction. The resulting coalgebra $\beta^\sharp : \mathcal{S}_{\mathtt{PROP}}(\Sigma) \to \mathcal{F}\mathcal{S}_{\mathtt{PROP}}(\Sigma)$ assigns to each diagram of $\mathcal{S}_{\mathtt{PROP}}(\Sigma)$ the set of transitions specified by the combined operational semantics of Figs. 2 and 3. The preceding discussion implies that, when e.g. $\mathsf{R} = \mathbb{N}$, bisimilarity for the Petri nets of [6] is a congruence.

5.2 Bialgebraic Semantics for Carboni-Walters Props

In this section we shall see two different ways of extending the GSOS specification of § 5.1 for CW props (see § 4.4). They correspond to the operational semantics of the black and white (co)monoids as given in Fig. 2. In the next section, we will see that these two different extensions give rise to two classic forms of synchronisation: à la Hoare and à la Milner.

Black distributive law. The first interprets the operations of the Frobenius structure as label synchronisation: from the black node derivations on the left of Fig. 2 we get GSOS specifications given by natural transformations $\text{—●<} \Rightarrow \mathcal{F}(\text{—●<}^\dagger)$, $\text{—●} \Rightarrow \mathcal{F}(\text{—●}^\dagger)$, $\text{>●—} \Rightarrow \mathcal{F}(\text{>●—}^\dagger)$, and $\text{●—} \Rightarrow \mathcal{F}(\text{●—}^\dagger)$. Recall that, here, we use the diagrams to denote their associated functors $\mathsf{Sig} \to \mathsf{Sig}$. By taking the coproduct of these and λ, the GSOS specification for props from § 5.1, we obtain a specification $\lambda_\bullet$ for $\mathcal{S}_{\mathtt{FR}}$. It is straightforward to verify that $\lambda_\bullet^\dagger : \mathcal{S}_{\mathtt{FR}}^\dagger \mathcal{F} \Rightarrow \mathcal{F}\mathcal{S}_{\mathtt{FR}}^\dagger$ preserves the equations of special Frobenius bimonoids (Fig. 4), yielding a distributive law $\lambda^\dagger_{\bullet/\mathsf{CW}} : \mathcal{S}_{\mathtt{CW}}^\dagger \mathcal{F} \Rightarrow \mathcal{F}\mathcal{S}_{\mathtt{CW}}^\dagger$. As before, with $\lambda^\dagger_{\bullet/\mathsf{CW}}$ we obtain a bialgebra $\beta_\bullet^\sharp : \mathcal{S}_{\mathtt{CW}}(\Sigma) \to \mathcal{F}\mathcal{S}_{\mathtt{CW}}(\Sigma)$ from any coalgebra $\beta : \Sigma \to \mathcal{F}\mathcal{S}_{\mathtt{CW}}(\Sigma)$.

White distributive law. When the set of labels A is an *Abelian group*, it is possible to give a different GSOS specifications for the Frobenius structure, capturing the group operation of A: from the white node derivations on the right of Fig. 2 we get GSOS specifications $\text{—○<} \Rightarrow \mathcal{F}(\text{—○<}^\dagger)$, $\text{—○} \Rightarrow \mathcal{F}(\text{—○}^\dagger)$, $\text{>○—} \Rightarrow \mathcal{F}(\text{>○—}^\dagger)$, and $\text{○—} \Rightarrow \mathcal{F}(\text{○—}^\dagger)$. Using a now familiar procedure we obtain a GSOS specifications $\lambda_\circ$ for $\mathcal{S}_{\mathtt{FR}}$. The group structure on A guarantees [39] that $\lambda_\circ^\dagger : \mathcal{S}_{\mathtt{FR}}^\dagger \mathcal{F} \Rightarrow \mathcal{F}\mathcal{S}_{\mathtt{FR}}^\dagger$ preserves the equations of special Frobenius bimonoids (Fig. 4). Therefore we get a distributive law $\lambda^\dagger_{\circ/\mathsf{CW}} : \mathcal{S}_{\mathtt{CW}}^\dagger \mathcal{F} \Rightarrow \mathcal{F}\mathcal{S}_{\mathtt{CW}}^\dagger$.

▶ Remark 10. Given the results in this section, one could ask if bialgebraic semantics works for *any* categorical structure. A notable case in which it fails is that of Lawvere theories [29]. These can be seen as props with a *natural* comonoid structure on each object [12]. One may define a monad for Lawvere theories following the same recipe as above. However, it turns out that this monad is incompatible with the GSOS specification for the comonoid given in Fig. 2. To see the problem consider a term d that can perform two transitions nondeterministically: $d \xrightarrow[a]{\epsilon} d$ and $d \xrightarrow[b]{\epsilon} d$. The naturality of the comonoid forces $d\,;\text{—●<} \approx d \oplus d$ but $d\,;\text{—●<}$ can only perform the $a\,a$ and $b\,b$ transitions while $d \oplus d$ can also perform $a\,b$ or $b\,a$. Thus the specification would not be compositional. For more details, we refer the reader to the appendix of [7].

6 Black and White Frobenius as Hoare and Milner Synchronisation

The role of this section is twofold: on the one hand we demonstrate how classical process calculus syntax benefits from a string diagrammatic treatment; on the other we draw attention towards a surprising observation, namely that the black and white Frobenius structures discussed previously provide the synchronisation mechanism of, respectively, CSP and CCS.

6.1 Syntax

We consider a minimal process calculus for simplicity. Assume a countable set $\mathcal{N}$ of *names*, a_1, a_2, …and a set $\mathcal{V}$ of *process variables*, $\mathtt{f}$, $\mathtt{g}$, …, equipped with a function $ar\colon \mathcal{V} \to \mathbb{N}$ that assigns the set of names that the process may use: $ar(\mathtt{f}) = n$ means that the process $\mathtt{f}$ uses only names $\{a_1, \dots, a_n\}$. This is Hoare's [28] notion of *alphabet* for process variables.

Roughly speaking, in a string diagram, dangling wires perform the job of variables. To ease the translation of terms to diagrams, we include permutations of names in the syntax, hereafter denoted by σ. For a permutation $\sigma\colon \mathcal{N} \to \mathcal{N}$, its support is the set $supp(\sigma) = \{a_i \mid a_i \neq \sigma(a_i)\}$; σ is *finitely supported* if $supp(\sigma)$ is finite. For each finitely supported permutation σ its *degree* is defined as the greatest $i \in \mathbb{N}$ such that $a_i \in supp(\sigma)$.

The set of processes is defined recursively as follows

$$P := P|P, \;\; \nu a_i(P), \;\; \mathtt{f}, \;\; P\sigma$$

where $a_i \in \mathcal{N}$, $\mathtt{f} \in \mathcal{V}$ and σ is a finitely supported permutation of names. The symbol $|$ stands for the parallel composition of processes. The symbol νa_i stands for the restriction, or hiding, of the name a_i. Observe that there are no primitives for prefixes, non-deterministic choice or recursion: these will appear in the declaration of process variables which we will describe in § 6.2. The idea here is to separate the behaviour, specified in the declaration of process variables, and the communication topology of the network, given by the syntax above. The notion of alphabet can be defined for all processes as follows:

$$al(P|Q) = al(P) \cup al(Q) \quad al(\nu a_i(P)) = al(P) \setminus \{a_i\} \quad al(\mathtt{f}) = \{a_1, \dots, a_{ar(\mathtt{f})}\} \quad al(P\sigma) = \sigma[al(P)]$$

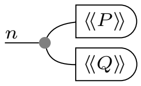
From one-dimensional to two-dimensional syntax. We use a typing discipline to guide the translation of terms to string diagrams:

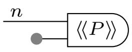
$$\frac{n \vdash P \quad n \vdash Q}{n \vdash P|Q} \qquad \frac{n+1 \vdash P}{n \vdash \nu a_{n+1}(P)} \qquad \frac{ar(\mathtt{f}) = n}{n \vdash \mathtt{f}} \qquad \frac{n \vdash P \quad degree(\sigma) \leq n}{n \vdash P\sigma} \qquad \frac{n \vdash P}{n+1 \vdash P} \tag{11}$$

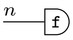
The meaning of the types is explained by the following lemma, easily proven by induction.

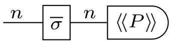
▶ **Lemma 11.** *If $n \vdash P$ then $al(P) \subseteq \{a_1, \dots a_n\}$.*

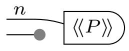
We will translate processes to the CW prop freely generated from $\Sigma = \{\mathtt{f}\colon (n, 0) \mid \mathtt{f} \in \mathcal{V}$ and $ar(\mathtt{f}) = n\}$; in particular a typed process $n \vdash P$ results in a string diagram of $\mathcal{S}_{\mathtt{CW}}(\Sigma)(n, 0)$. The translation $\langle\!\langle \cdot \rangle\!\rangle$ is defined recursively on typed terms as follows:

$\langle\!\langle n \vdash P|Q \rangle\!\rangle =$ 

$\langle\!\langle n \vdash \nu a_{n+1}(P) \rangle\!\rangle =$ 

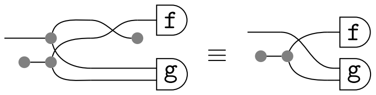
$\langle\!\langle n \vdash \mathtt{f} \rangle\!\rangle =$ 

$\langle\!\langle n \vdash P\sigma \rangle\!\rangle =$ 

$\langle\!\langle n+1 \vdash P \rangle\!\rangle =$ 

where for σ with $degree(\sigma) < n$, $\overline{\sigma}\colon n \to n$ is the obvious corresponding arrow in $\mathcal{S}_{\mathtt{CW}}(\Sigma)$.

▶ **Example 12.** Let $\mathcal{V} = \{\mathtt{f}, \mathtt{g}\}$ with $ar(\mathtt{f}) = 1$ and $ar(\mathtt{g}) = 2$. Let $[a_2/a_1]\colon \mathcal{N} \to \mathcal{N}$ be the permutation swapping a_1 and a_2. One can easily check that $1 \vdash \nu a_2(\mathtt{f}[a_2/a_1] \,|\, \mathtt{g})$. Then $\langle\!\langle 1 \vdash \nu a_2(\mathtt{f}[a_2/a_1] \,|\, \mathtt{g}) \rangle\!\rangle$ is as on the right.



6.2 Semantics

In order to give semantics to the calculus, we assume a set $\mathcal{A}$ of actions, α, β, ... Since, we will consider different sets of actions (for Hoare and Milner synchronisation), we assume them to be functions of type $\mathcal{N} \to M$ for some monoid $(M, +, 0)$. The support of an action α is the set $\{a_i \mid \alpha(a_i) \neq 0\}$. The alphabet of α, written $al(\alpha)$ is identified with its support.

For Hoare synchronisation, the monoid M is $(2, \cup, 0)$, while for Milner it is $(\mathbb{Z}, +, 0)$. In both cases, we will write a_i for the function mapping the name a_i to 1 and all the others to 0. For Milner synchronisation, write $\overline{a_i}$ for the function mapping a_i to -1.

To give semantics to processes, we need a *process declaration* for each $\mathtt{f} \in \mathcal{V}$. That is, an expression $\mathtt{f}{:=}\sum_{i\in I} \alpha_i.P_i$, for some finite set I, $\alpha_i \in \mathcal{A}$ and processes P_i such that

$$\{a_1, \ldots a_{ar(\mathtt{f})}\} \subseteq \bigcup_{i\in I} al(\alpha_i) \cup \bigcup_{i\in I} al(P_i) \qquad (12)$$

The basic behaviour of process declarations is captured by the three rules below.

$$\frac{}{\mathtt{f} \xrightarrow{0} \mathtt{f}} \qquad \frac{\mathtt{f}{:=}\sum_{i\in I} \alpha_i.P_i}{\mathtt{f} \xrightarrow{\alpha_i} P_i} \qquad \frac{P \xrightarrow{\alpha} P'}{P\sigma \xrightarrow{\alpha\circ\sigma} P'\sigma} \qquad (13)$$

▶ **Example 13.** Recall $\mathtt{f}$ and $\mathtt{g}$ from Example 12. Assume declarations $\mathtt{f}{:=}a_1.\nu a_2(\mathtt{f}[a_2/a_1] \mid \mathtt{g})$ and $\mathtt{g}{:=}a_1.\mathtt{g} + a_2.\mathtt{g}$. Observe that they respect (12). We have that $\mathtt{g} \xrightarrow{a_1} \mathtt{g}$ and $\mathtt{g} \xrightarrow{a_2} \mathtt{g}$ while $\mathtt{f} \xrightarrow{a_1} \nu a_2(\mathtt{f}[a_2/a_1] \mid \mathtt{g})$. Similarly $\mathtt{f}[a_2/a_1] \xrightarrow{a_2} (\nu a_2(\mathtt{f}[a_2/a_1] \mid \mathtt{g}))[a_2/a_1]$.

To define the semantics of parallel and restriction, we need to distinguish between the Hoare and Milner synchronisation patterns.

Hoare synchronisation. Here actions are functions $\alpha\colon \mathcal{N} \to 2$, which can equivalently be thought of as subsets of $\mathcal{N}$. The synchronisation mechanism presented below is analogous to the one used in CSP [28]. The main difference is the level of concurrency: the classical semantics [28] is purely interleaving, while for us it is a step semantics. Essentially, in $P|Q$, the processes P and Q may evolve independently on the non-shared names, i.e. the evolution of two or more processes may happen at the same time. It is for this reason that our actions are sets of names. The operational semantics of parallel and restriction is given by rules

$$\frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\beta} Q' \quad \alpha \cap al(Q) = \beta \cap al(P)}{P|Q \xrightarrow{\alpha\cup\beta} P'|Q'} \qquad \frac{P \xrightarrow{\alpha} P'}{\nu a_i(P) \xrightarrow{\alpha\setminus\{a_i\}} \nu a_i(P')} \qquad (14)$$

We write $\xrightarrow{\alpha}_H$ for the transition systems generated by the rules (13), (14). By a simple inductive argument, using (12) as base case, we see that for all processes P, if $P \xrightarrow{\alpha} P'$ then $\alpha \subseteq al(P)$. The rule for parallel, therefore, ensures that P and Q synchronise over all of their shared names. The rule for restriction hides a_i from the environment. For instance, if $\alpha = \{a_i\}$, then $\nu a_i(P) \xrightarrow{\emptyset} \nu a_i(P')$. If $\alpha = \{a_j\}$ with $a_j \neq a_i$, then $\nu a_i(P) \xrightarrow{\{a_j\}} \nu a_i(P')$.

▶ **Example 14.** Recall $\mathtt{f}$ and $\mathtt{g}$ from Example 13. We have that $\mathtt{f} \xrightarrow{a_1}_H \nu a_2(\mathtt{f}[a_2/a_1] \mid \mathtt{g})$. From $\nu a_2(\mathtt{f}[a_2/a_1] \mid \mathtt{g})$, there are two possibilities: either $\mathtt{f}[a_2/a_1]$ and $\mathtt{g}$ synchronise on a_2, and in this case we have $\nu a_2(\mathtt{f}[a_2/a_1] \mid \mathtt{g}) \xrightarrow{\emptyset}$, or $\mathtt{g}$ proceeds without synchronising on a_1, therefore $\nu a_2(\mathtt{f}[a_2/a_1] \mid \mathtt{g}) \xrightarrow{\{a_1\}}_H$ since a_1 belongs to $al(\mathtt{g})$ and not to $al(\mathtt{f}[a_2/a_1])$.

Milner synchronisation. We take $\mathcal{A} = \mathbb{Z}^{\mathcal{N}}$. Sum of functions, denoted by $+$, is defined pointwise and we write 0 for its unit, the constant 0 function.

$$\frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\beta} Q'}{P|Q \xrightarrow{\alpha+\beta} P'|Q'} \qquad \frac{P \xrightarrow{\alpha} P'}{\nu a_i(P) \xrightarrow{\alpha} \nu a_i(P')}\,\alpha(a_i) = 0 \tag{15}$$

We write $\xrightarrow{\alpha}_M$ for the transition system generated by the rules (13), (15).

Functions in $\mathbb{Z}^{\mathcal{N}}$ to represent concurrent occurrences of CCS send and receive actions. A single CCS action a is the function mapping a to 1 and all other names to 0. Similarly, the action $\bar{a}$ maps a to -1 and the other names to 0. The silent action τ is the function 0. With this in mind, it is easy to see that, similarly to CCS, the rightmost rule forbids $\nu a_i(P) \xrightarrow{\alpha} \nu a_i(P')$ whenever $\alpha = a_i$ or $\alpha = \bar{a}_i$. CCS-like synchronisation is obtained by the leftmost rule: when $\alpha = a_i$ and $\beta = \bar{a}_i$, one has that $P|Q \xrightarrow{0} P'|Q'$.

A simple inductive argument confirms that $P \xrightarrow{0} P$ for any process P. Then, by the leftmost rule again, one has that whenever $Q \xrightarrow{\beta} Q'$, then $P|Q \xrightarrow{\beta} P|Q'$. Note, however, that as in § 6.2, while our synchronisation mechanism is essentially Milner's CSS handshake, our semantics is not interleaving and allows for step concurrency. It is worth remarking that the operational rules in (15) have already been studied by Milner in its work on SCCS [37].

Semantic correspondence. For an action $\alpha \colon \mathcal{N} \to M$ with $al(\alpha) \subseteq \{a_1, \ldots a_n\}$, we write $n \vdash \alpha$ for the restriction $\{a_1, \ldots, a_n\} \to M$. Define coalgebras $\beta_b, \beta_w : \Sigma \to \overline{\mathcal{P}_\kappa}(L\,;\mathcal{S}_{\texttt{CW}}(\Sigma)\,; L)$ for each $\texttt{f} \in \Sigma_{n,0}$ where $\texttt{f}{:=}\sum_{i \in I} \alpha_i.P_i$ as

$$\beta_b(\texttt{f}) = \beta_w(\texttt{f}) = \left\{\left(n \vdash \alpha_i, \langle\!\langle P_i \rangle\!\rangle, \bullet\right) \mid i \in I\right\} \cup \left\{\left(n \vdash 0, \texttt{f}, \bullet\right)\right\}.$$

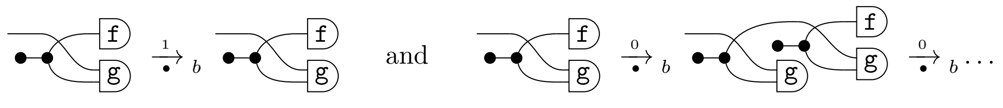
For both β_b and β_w, L is the span $\mathbb{N} \xleftarrow{|\cdot|} A^* \xrightarrow{|\cdot|} \mathbb{N}$, but $A = 2$ for β_b and $A = \mathbb{Z}$ for β_w.

Via the distributive law (§ 5.2) for the black Frobenius, we obtain the coalgebra $\beta_b^\sharp \colon \mathcal{S}_{\texttt{CW}}(\Sigma) \to \overline{\mathcal{P}_\kappa}(L\,;\mathcal{S}_{\texttt{CW}}(\Sigma)\,;L)$. Via the white Frobenius, we obtain $\beta_w^\sharp \colon \mathcal{S}_{\texttt{CW}}(\Sigma) \to \overline{\mathcal{P}_\kappa}(L\,;\mathcal{S}_{\texttt{CW}}(\Sigma)\,;L)$. We write $c \xrightarrow[\alpha]{\beta}{}_b d$ for $(\alpha, \beta, d) \in \beta_b^\sharp(c)$ and $c \xrightarrow[\alpha]{\beta}{}_w d$ for $(\alpha, \beta, d) \in \beta_w^\sharp(c)$. The correspondence can now be stated formally.

▶ **Theorem 15.** *Let $n \vdash P$ and $n \vdash \alpha$ such that $al(\alpha) \subseteq al(P)$.*

- ***Hoare is black.*** *If $P \xrightarrow{\alpha}_H P'$ then $\overset{n}{\text{—}}\boxed{\langle\!\langle P \rangle\!\rangle} \xrightarrow[\bullet]{n\vdash\alpha}{}_b \overset{n}{\text{—}}\boxed{\langle\!\langle P' \rangle\!\rangle}$. Vice versa, if $\overset{n}{\text{—}}\boxed{\langle\!\langle P \rangle\!\rangle} \xrightarrow[\bullet]{n\vdash\alpha}{}_b \overset{n}{\text{—}}\boxed{d}$ then there is $n \vdash P'$ s.t. $P \xrightarrow{\alpha}_H P'$ and $\overset{n}{\text{—}}\boxed{\langle\!\langle P' \rangle\!\rangle} = \overset{n}{\text{—}}\boxed{d}$.*
- ***Milner is white.*** *If $P \xrightarrow{\alpha}_M P'$ then $\overset{n}{\text{—}}\boxed{\langle\!\langle P \rangle\!\rangle} \xrightarrow[\bullet]{n\vdash\alpha}{}_w \overset{n}{\text{—}}\boxed{\langle\!\langle P' \rangle\!\rangle}$. Vice versa, if $\overset{n}{\text{—}}\boxed{\langle\!\langle P \rangle\!\rangle} \xrightarrow[\bullet]{n\vdash\alpha}{}_w \overset{n}{\text{—}}\boxed{d}$ then there is $n \vdash P'$ s.t. $P \xrightarrow{\alpha}_M P'$ and $\overset{n}{\text{—}}\boxed{\langle\!\langle P' \rangle\!\rangle} = \overset{n}{\text{—}}\boxed{d}$.*

▶ **Example 16.** We illustrate the semantic correspondence by returning to Example 13. Diagrammatically, it yields the following transitions:



7 Related and Future Work

The terminology *Hoare and Milner synchronisation* is used in Synchronised Hyperedge Replacement (SHR) [20, 33]. Our work is closely related to SHR: indeed, the prop $\mathcal{S}_{\mathtt{CW}}(\Sigma)$ has arrows open hypergraphs, where hyperedges are labeled with elements of Σ [5]. To define a coalgebra $\beta\colon \Sigma \to \mathcal{F}\mathcal{S}_{\mathtt{CW}}(\Sigma)$ is to specify a transition system for each label in Σ. Then, constructing the coalgebra $\beta^{\sharp}\colon \mathcal{S}_{\mathtt{CW}}(\Sigma) \to \mathcal{F}\mathcal{S}_{\mathtt{CW}}(\Sigma)$ from a distributive law amounts to giving a transition system to all hypergraphs according to some synchronisation policy (e.g. à la Hoare or à la Milner). SHR systems equipped with Hoare and Milner synchronisation are therefore instances of our approach. A major difference is our focus on the algebraic aspects: e.g. since string diagrams can be regarded as syntax as well as combinatorial entities, their syntactic nature allows for the bialgebraic approach, and simple inductive proofs. The operational rules in Figure 3 are also those of tile systems [23]. However, in the context of tiles, transitions are arrows of the vertical *category*: this forces every state to perform at least one identity transition. For example, it is not possible to consider empty sets of transitions, which can be a useful feature in the string diagrammatic approach, see [8].

Amongst the many other related models, it is worth mentioning bigraphs [38]. While also graphical, bigraphs can be nested hierarchically, a capability that we have not considered. Moreover, the behaviour functor $\mathcal{F}$ in § 5 forces the labels and the arriving states to have the same sort as the starting states. Therefore, fundamental mobility mechanisms such as scope-extrusion cannot immediately be addressed within our framework. We are confident, however, that the solid algebraic foundation we have laid here for the operational semantics of two-dimensional syntax will be needed to shed light on such concepts as hierarchical composition and mobility. Some ideas may come from [14].

References

1 Samson Abramsky and Bob Coecke. A Categorical Semantics of Quantum Protocols. In *LICS 2004*, pages 415–425, 2004. doi:10.1109/LICS.2004.1319636.

2 John Baez and Jason Erbele. Categories In Control. *Theory Appl. Categ.*, 30:836–881, 2015. arXiv:1405.6881.

3 John Baez and Brendan Fong. A compositional framework for passive linear circuits. *J. Complex Netw.*, 54:5, 2015.

4 Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *J. ACM*, 42(1):232–268, 1995.

5 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. Rewriting modulo symmetric monoidal structure. In *LICS 2016*, pages 1–10, 2016.

6 Filippo Bonchi, Joshua Holland, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. Diagrammatic Algebra: from Linear to Concurrent Systems. In *POPL 2019*, page 25, 2019.

7 Filippo Bonchi, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. Bialgebraic Semantics for String Diagrams. *CoRR*, 2019. arXiv:1906.01519.

8 Filippo Bonchi, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. Graphical Affine Algebra. In *To appear in LICS 2019*, 2019.

9 Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. A Categorical Semantics of Signal Flow Graphs. In *CONCUR 2014*, pages 435–450, 2014.

10 Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. Full Abstraction for Signal Flow Graphs. In *POPL 2015*, pages 515–526, 2015.

11 Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. The Calculus of Signal Flow Diagrams I: Linear relations on streams. *Inf. Comput.*, 252:2–29, 2017. doi:10.1016/j.ic.2016.03.002.

12 Filippo Bonchi, Pawel Sobociński, and Fabio Zanasi. Deconstructing Lawvere with distributive laws. *J. Log. Algebr. Meth. Program.*, 95:128–146, 2018. doi:10.1016/j.jlamp.2017.12.002.

13 Marcello M. Bonsangue, Helle Hvid Hansen, Alexander Kurz, and Jurriaan Rot. Presenting Distributive Laws. In *Algebra and Coalgebra in Computer Science - 5th International Conference, CALCO 2013, Warsaw, Poland, September 3-6, 2013. Proceedings*, pages 95–109, 2013.

14 Roberto Bruni, Ugo Montanari, Gordon D. Plotkin, and Daniele Terreni. On Hierarchical Graphs: reconciling Bigraphs, Gs-monoidal Theories and Gs-graphs. *Fundam. Inform.*, 134(3–4):287–317, 2014.

15 Maria Grazia Buscemi and Ugo Montanari. A First Order Coalgebraic Model of pi-Calculus Early Observational Equivalence. In Lubos Brim, Petr Jancar, Mojmír Kretínský, and Antonín Kucera, editors, *CONCUR 2002 - Concurrency Theory, 13th International Conference, Brno, Czech Republic, August 20-23, 2002, Proceedings*, volume 2421 of *Lecture Notes in Computer Science*, pages 449–465. Springer, 2002. doi:10.1007/3-540-45694-5_30.

16 Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000.

17 Giuseppe Castagna, Jan Vitek, and Francesco Zappa Nardelli. The Seal Calculus. *Inform. Comput.*, 201(1):1–54, 2005.

18 Bob Coecke and Ross Duncan. Interacting Quantum Observables. In *ICALP 2008*, pages 298–310, 2008.

19 Robert De Simone. Higher-level synchronising devices in Meije-SCCS. *Theor. Comput. Sci.*, 37:245–267, 1985.

20 Pierpaolo Degano and Ugo Montanari. A model for distributed systems based on graph rewriting. *J. ACM*, 34(2):411–449, 1987.

21 Brendan Fong, Paolo Rapisarda, and Paweł Sobociński. A categorical approach to open and interconnected dynamical systems. In *LICS 2016*, pages 1–10, 2016.

22 Brendan Fong and David I. Spivak. Hypergraph Categories. *CoRR*, abs/1806.08304, 2018. arXiv:1806.08304.

23 Fabio Gadducci and Ugo Montanari. The tile model. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 133–166. MIT Press, 2000.

24 Dan R. Ghica. Diagrammatic Reasoning for Delay-Insensitive Asynchronous Circuits. In *Abramsky Festschrift*, pages 52–68, 2013. doi:10.1007/978-3-642-38164-5_5.

25 Jan Friso Groote. Transition system specifications with negative premises. *Theor. Comput. Sci.*, 118(2):263–299, 1993.

26 David Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, 1987.

27 C. A. R. Hoare. Communicating Sequential Processes. *Commun. ACM*, 21(8):666–677, August 1978.

28 C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, Inc., 1985.

29 Martin Hyland and John Power. Lawvere theories and monads. In *Computation, Meaning, and Logic*, pages 437–458, 2007.

30 Kurt Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*, volume 1. Springer Science & Business Media, 2013.

31 Piergiulio Katis, Nicoletta Sabadini, and Robert Frank Carslaw Walters. Span(Graph): an algebra of transition systems. In *AMAST 1997*, volume 1349 of *LNCS*, pages 322–336. Springer, 1997. doi:10.1007/bfb0000479.

32 Bartek Klin. Bialgebras for structural operational semantics: an introduction. *Theor. Comput. Sci.*, 412(38):5043–5069, 2011.

33 Ivan Lanese and Ugo Montanari. Hoare vs Milner: Comparing Synchronizations in a Graphical Framework With Mobility. *Electr. Notes Theor. Comput. Sci.*, 154(2):55–72, 2006. doi:10.1016/j.entcs.2005.03.032.

34 Marina Lenisa, John Power, and Hiroshi Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. *Electr. Notes Theor. Comput. Sci.*, 33:230–260, 2000.

35 Saunders Mac Lane. Categorical Algebra. *B. Am. Math. Soc.*, 71:40–106, 1965.

36 Robin Milner. *A Calculus of Communicating Systems*. Springer-Verlag, 1982.

37 Robin Milner. Calculi for synchrony and asynchrony. *Theor. Comput. Sci.*, 25(3):267–310, 1983.

38 Robin Milner. *The space and motion of communicating agents*. Cambridge University Press, 2009.

39 Dusko Pavlovic. Quantum and classical structures in nondeterministic computation. In *QI 2009*, pages 143–157, 2009.

40 Wolfgang Reisig. *Petri nets: an introduction*, volume 4. Springer Science & Business Media, 2012.

41 Jurriaan Rot. *Enhanced Coinduction*. PhD thesis, University of Leiden, 2015.

42 Davide Sangiorgi and David Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

43 Peter Selinger. A survey of graphical languages for monoidal categories. *Springer Lecture Notes in Physics*, 13(813):289–355, 2011.

44 Paweł Sobociński. A non-interleaving process calculus for multi-party synchronisation. In *ICE 2009*, pages 87–98, 2009. doi:10.4204/EPTCS.12.6.

45 Daniele Turi and Gordon Plotkin. Towards a mathematical operational semantics. In *LICS 1997*, pages 280–291, 1997.

A Sound Algorithm for Asynchronous Session Subtyping

Mario Bravetti
Department of Computer Science / INRIA FoCUS Team, University of Bologna, Italy

Marco Carbone
IT University of Copenhagen, Denmark

Julien Lange
University of Kent, UK

Nobuko Yoshida
Imperial College London, UK

Gianluigi Zavattaro
Department of Computer Science / INRIA FoCUS Team, University of Bologna, Italy

Abstract

Session types, types for structuring communication between endpoints in distributed systems, are recently being integrated into mainstream programming languages. In practice, a very important notion for dealing with such types is that of subtyping, since it allows for typing larger classes of system, where a program has not precisely the expected behavior but a similar one. Unfortunately, recent work has shown that subtyping for session types in an asynchronous setting is undecidable. To cope with this negative result, the only approaches we are aware of either restrict the syntax of session types or limit communication (by considering forms of bounded asynchrony). Both approaches are too restrictive in practice, hence we proceed differently by presenting an algorithm for checking subtyping which is sound, but not complete (in some cases it terminates without returning a decisive verdict). The algorithm is based on a tree representation of the coinductive definition of asynchronous subtyping; this tree could be infinite, and the algorithm checks for the presence of finite witnesses of infinite successful subtrees. Furthermore, we provide a tool that implements our algorithm and we apply it to many examples that cannot be managed with the previous approaches.

2012 ACM Subject Classification Theory of computation → Concurrency

Keywords and phrases Session types, Concurrency, Subtyping, Algorithm

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.38

Related Version A full version of this paper is available at `https://arxiv.org/abs/1907.00421`.

Funding H2020-MSCA-RISE Project 778233 (BEHAPI); EPSRC EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1, and EP/N028201/1.

1 Introduction

Session types are behavioural types that specify the structure of communication between the endpoints of a distributed system or the processes of a concurrent program. In recent years, session types have been integrated into several mainstream programming languages (see, e.g., [4, 19, 25, 28–30, 32]) where they specify the pattern of interactions that each endpoint must follow, i.e., a communication protocol. The notion of duality is at the core of theories based on session types, where it guarantees that each send (resp. receive) action is matched by a corresponding receive (resp. send) action, and thus rules out deadlocks and orphan messages. A two-party communication protocol specified as a pair of session types is "correct" (deadlock free, etc) when these types are dual of each other. Unfortunately, in

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 38; pp. 38:1–38:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

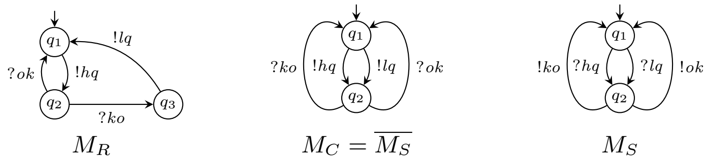


Figure 1 Video streaming example. M_R is the (refined) session type of the client, M_C is a supertype of M_R, and M_S is the session type of the server.

practice, duality is a too strict prerequisite, since it does not provide programmers with the flexibility necessary to build practical implementations of a given protocol. A natural solution for relaxing this rigid constraint is to adopt a notion of (session) subtyping which lets programmers implement refinements of the specification (given as a session type). In particular, an endpoint implemented as program P_2 with type M_2 can always be safely replaced by another program P_1 with type M_1 whenever M_1 is a subtype of M_2 (written $M_1 \preccurlyeq M_2$ in this paper).

The two main known notions of subtyping for session types differ in the type of communication they support: either synchronous (rendez-vous) or asynchronous (over unbounded FIFO channels). *Synchronous session subtyping* only allows for a subtype to implement fewer internal choices (sends), and more external choices (receives), than its supertype. Hence checking whether two types are related can be done efficiently (quadratic time wrt. the size of the types [23]). Synchronous session subtyping is of limited interest in modern programming languages such as Go and Rust, which provide *asynchronous* communication over channels. Indeed, in an asynchronous setting, the programmer needs to be able to make the best of the flexibility given by non-blocking send actions. This is precisely what the *asynchronous session subtyping* offers: it widens the synchronous subtyping relation by allowing the subtype to anticipate send actions, when this does not affect its communication partner. We illustrate the salient points of the asynchronous session subtyping with Figure 1, which depicts the hypothetical session types of the client and server endpoints of a video streaming service, represented as communicating machines – an equivalent formalism [6, 13]. Machine M_S (right) is a server which can deal with two types of requests: it can receive either a message *lq* (low-quality) or a message *hq* (high-quality). After receiving a message of either type, the server replies with *ok* or *ko*, indicating whether the request can be fulfilled, then it returns to its starting state. Machine M_C (middle) represents the type of the client. It is the *dual* of the server M_S (written $\overline{M_S}$), as required in standard two-party session types without subtyping. A programmer may want to implement a slightly improved program which behaves as Machine M_R (left). This version requests high-quality (*hq*) streaming first, and falls back to low-quality (*lq*) if the request is denied (it received *ko*). In fact, machine M_R is an (asynchronous) subtype of machine M_C. Indeed, M_R is able to receive the same set of messages as M_C, each of the sent messages are also allowed by M_C, and the system consisting of the parallel composition of machines M_R and M_S is free from deadlocks and orphan messages. We will use this example in the rest of the paper to illustrate our theory.

Recently, we have proven that checking whether two types are in the asynchronous subtyping relation is, unfortunately, *undecidable* [7, 8, 24]. In order to mitigate this negative result, some theoretical algorithms have been proposed for restricted subclasses of session types. These restrictions can be divided into two main categories: syntactical restrictions, i.e., allowing only one type of non-unary branching (internal or external), or adding bounds to the size of the FIFO communication channels. Both types of restrictions are problematic in practice. Syntactic restrictions disallow protocols featuring both types of internal/external choices, e.g., the machines M_C and M_S in Figure 1 contain (non-unary) external and internal choices. On the other hand, applying a bound to the subtyping relation is generally difficult because (i) it may be undecidable whether such a bound exists, (ii) the channel bounds used in the implementation (if any) might not be known at compile time, and (iii) very simple systems, such as the one in Figure 1, require unbounded communication channels.

In this paper, we give an algorithm that can soundly deal with a much larger class of session types. Rather than imposing syntactical restrictions or bounds, we describe an algorithm whose termination condition is based on a well-quasi order between pairs of candidate subtypes. This condition allows us to construct a finite tree representation of the coinductive definition of asynchronous subtyping, which we use to synthesise intermediate automata. Finally, we give a sufficient condition for asynchronous subtyping based on a compatibility relation between these intermediate automata. This compatibility check is similar to that of subtyping for recursive types [3,16,21,23]. We provide a full implementation of our algorithm and show that it performs well on examples taken from the literature.

2 Communicating Machines and Asynchronous Subtyping

In this section, we recall the definition of two-party communicating machines, that communicate over unbounded FIFO channels (§ 2.1), and define asynchronous subtyping for session types [10, 11], which we adapt to communicating machines, following [8] (§ 2.2).

2.1 Communicating Machines

Let $\mathbb{A}$ be a (finite) alphabet, ranged over by a, b, etc. We let ω, ω', etc. range over words in $\mathbb{A}^*$. The set of send (resp. receive) actions is $Act_! = \{!\} \times \mathbb{A}$, (resp. $Act_? = \{?\} \times \mathbb{A}$). The set of actions is $Act = Act_! \cup Act_?$, ranged over by ℓ, where send action $!a$ puts message a on an (unbounded) buffer, while receive action $?a$ represents the consumption of a from a buffer. We define $dir(!a) \overset{\text{def}}{=} !$ and $dir(?a) \overset{\text{def}}{=} ?$ and let ψ and φ range over Act^*. We write $\cdot$ for the concatenation operator on words.

In this work, we only consider communicating machines which correspond to (two-party) session types. Hence, we focus on deterministic (communicating) finite-state machines, without mixed states (i.e., states that can fire both send and receive actions) as in [13, 14].

▶ **Definition 1** (Communicating Machine). *A communicating machine M is a tuple (Q, q_0, δ) where Q is the (finite) set of states, $q_0 \in Q$ is the initial state, and $\delta \in Q \times Act \times Q$ is a transition relation such that $\forall q, q', q'' \in Q.\ \forall \ell, \ell' \in Act$: **(1)** $(q, \ell, q'), (q, \ell', q'') \in \delta$ implies $dir(\ell) = dir(\ell')$, and **(2)** $(q, \ell, q'), (q, \ell, q'') \in \delta$ implies $q' = q''$. We write $q \xrightarrow{\ell} q'$ for $(q, \ell, q') \in \delta$, omit unnecessary labels, and write $\rightarrow^*$ for the reflexive transitive closure of $\rightarrow$.*

Condition **(1)** enforces directed states, while Condition **(2)** enforces determinism.

Given $M = (Q, q_0, \delta)$, we say that $q \in Q$ is *final*, written $q \not\rightarrow$, iff $\forall q' \in Q.\ \forall \ell \in Act.\ (q, \ell, q') \notin \delta$. A state $q \in Q$ is *sending* (resp. *receiving*) iff q is not final and $\forall q' \in Q.\ \forall \ell \in Act.\ (q, \ell, q') \in \delta.\ dir(\ell) = !$ (resp. $dir(\ell) = ?$). We write $\delta(q, \ell)$ for q' iff $(q, \ell, q') \in \delta$.

We write $q_0 \xrightarrow{\ell_1 \cdots \ell_k} q_k$ iff there are $q_1, \dots, q_{k-1} \in Q$ such that $q_{i-1} \xrightarrow{\ell_i} q_i$ for $1 \leq i \leq k$. Given a list of messages $\omega = a_1 \cdots a_k$ $(k \geq 0)$, we write $?\omega$ for the list $?a_1 \cdots ?a_k$ and $!\omega$ for $!a_1 \cdots !a_k$. We write $q \xrightarrow{!}{}^* q'$ iff $\exists \omega \in \mathbb{A}.\ q \xrightarrow{!\omega} q'$ and $q \xrightarrow{?}{}^* q'$ iff $\exists \omega \in \mathbb{A}.\ q \xrightarrow{?\omega} q'$ (note that ω may be empty, in which case $q = q'$). Given $\psi \in Act^*$ we write $\mathsf{snd}(\psi)$ (resp. $\mathsf{rcv}(\psi)$) for the largest sub-sequence of ψ which consists only of the messages of send (resp. receive) actions.

2.2 Asynchronous Session Subtyping

Input trees and contexts. In order to formalise the subtyping relation, we use syntactic constructs used to record the input actions that have been anticipated by a candidate supertype, e.g., machine M_2 in Definition 5, as well as the local states it may reach.

▶ **Definition 2** (Input Tree). *An input tree is a term of the grammar:* $T \ ::= \ q \ \mid \ \langle a_i : T_i \rangle_{i \in I}$.

In the sequel, we use $\mathcal{T}_Q$ to denote the input trees over states $q \in Q$. An input context is an input tree with a "hole" in the place of a sub-term.

▶ **Definition 3** (Input Context). *An input context is a term of* $\mathcal{A} \ ::= \ [\,]_j \ \mid \ \langle a_i : \mathcal{A}_i \rangle_{i \in I}$, *where all indices* j*, denoted by* $I(\mathcal{A})$*, are distinct and are associated to holes.*

For input trees and contexts of the form $\langle a_i : T_i \rangle_{i \in I}$ and $\langle a_i : \mathcal{A}_i \rangle_{i \in I}$, we assume that $I \neq \emptyset$, $\forall i \neq j \in I.\ a_i \neq a_j$, and that the order of the sub-terms is irrelevant. When convenient, we use set-builder notation to construct input trees or contexts, e.g., $\langle a_i : T_i \mid i \in I \rangle$.

Given an input context $\mathcal{A}$ and an input context $\mathcal{A}_i$ for each i in $I(A)$, we write $\mathcal{A}[\mathcal{A}_i]^{i \in I(A)}$ for the input context obtained by replacing each hole $[\,]_i$ in $\mathcal{A}$ by the input context $\mathcal{A}_i$. We write $\mathcal{A}[T_i]^{i \in I(A)}$ for the input tree where holes are replaced by input trees.

Auxiliary functions. In the rest of the paper we use the following auxiliary functions on communicating machines. Given a machine $M = (Q, q_0, \delta)$ and a state $q \in Q$, we define:

- $\mathsf{cycle}(\star, q) \iff \exists \omega \in \mathbb{A}^*, \omega' \in \mathbb{A}^+, q' \in Q.\ q \xrightarrow{\star\omega} q' \xrightarrow{\star\omega'} q'$ (with $\star \in \{!, ?\}$),
- $\mathsf{in}(q) = \{a \mid \exists q'.q \xrightarrow{?a} q'\}$ and $\mathsf{out}(q) = \{a \mid \exists q'.q \xrightarrow{!a} q'\}$,
- let the *partial* function $\mathsf{inTree}(\cdot)$ be defined as:

$$\mathsf{inTree}(q) = \begin{cases} \bot & \text{if } \mathsf{cycle}(?, q) \\ q & \text{if } \mathsf{in}(q) = \varnothing \\ \langle a_i : \mathsf{inTree}(\delta(q, ?a_i)) \rangle_{i \in I} & \text{if } \mathsf{in}(q) = \{a_i \mid i \in I\} \neq \varnothing \end{cases}$$

Predicate $\mathsf{cycle}(\star, q)$ says that, from q, we can reach a cycle with only sends (resp. receives), depending on whether $\star =!$ or $\star =?$. The function $\mathsf{in}(q)$ (resp. $\mathsf{out}(q)$) returns the messages that can be received (resp. sent) in q. When defined, $\mathsf{inTree}(q)$ returns the tree containing all sequences of messages which can be received from q until a final or sending state is reached. Intuitively, $\mathsf{inTree}(q)$ is undefined when $\mathsf{cycle}(?, q)$ as it would return an infinite tree.

▶ **Example 4.** Given M_C (Figure 1), we have $\mathsf{inTree}(q_1) = q_1$ and $\mathsf{inTree}(q_2) = \langle ok : q_1,\ ko : q_1 \rangle$.

Asynchronous subtyping. We present our definition of asynchronous subtyping (following the orphan-message-free version from [11]). Our definition is a simple adaptation[1] of [8, Definition 2.4] (given on syntactical session types) to the setting of communicating machines.

▶ **Definition 5** (Asynchronous Subtyping). *Let* $M_i = (Q_i, q_{0_i}, \delta_i)$ *for* $i \in \{1, 2\}$. $\mathcal{R}$ *is an asynchronous subtyping relation on* $Q_1 \times \mathcal{T}_{Q_2}$ *such that* $(p, T) \in \mathcal{R}$ *implies what follows:*

1. *if* $p \not\rightarrow$ *then* $T = q$ *such that* $q \not\rightarrow$*;*
2. *if* p *is a receiving state then*

[1] In definitions for syntactical session types, e.g., [26], input contexts are used to accumulate inputs that precede anticipated outputs; here, having no specific syntax for inputs, we use input trees instead.

a. *if $T = q$ then q is receiving and $\forall q' \in Q_2$ s.t. $q \xrightarrow{?a} q'$. $\exists p'$ s.t. $p \xrightarrow{?a} p' \wedge (p', q') \in \mathcal{R}$;*

b. *if $T = \langle a_i : T_i \rangle_{i \in I}$ then $\forall i \in I.\ \exists p'$ s.t. $p \xrightarrow{?a_i} p' \wedge (p', T_i) \in \mathcal{R}$;*

3. *if p is a sending state then*

a. *if $T = q$ then q is sending and $\forall p' \in Q_1$ s.t. $p \xrightarrow{!a} p'$. $\exists q'$ s.t. $q \xrightarrow{!a} q' \wedge (p', q') \in \mathcal{R}$ or*

b. *if $T = \mathcal{A}[q_i]^{i \in I}$ then let $\mathcal{A}_i[q_{i,h}]^{h \in H_i} = \mathsf{inTree}(q_i)$ and if $p \xrightarrow{!a} p'$ then $\forall i \in I. \forall h \in H_i.\ \exists q'_{i,h}.\ q_{i,h} \xrightarrow{!a} q'_{i,h} \wedge (p', \mathcal{A}[\mathcal{A}_i[q'_{i,h}]^{h \in H_i}]^{i \in I}) \in \mathcal{R}$ and, if $\mathcal{A}[\mathcal{A}_i[\,]^{h \in H_i}]^{i \in I}$ is not a single hole, then $\neg\mathsf{cycle}(!, p)$.*

M_1 is an asynchronous subtype of M_2, written $M_1 \preccurlyeq M_2$, if there is an asynchronous subtyping relation $\mathcal{R}$ such that $(q_{0_1}, q_{0_2}) \in \mathcal{R}$.

The relation $M_1 \preccurlyeq M_2$ checks that M_1 is a subtype of M_2 by executing M_1 and simulating its execution with M_2. M_1 may fire send actions earlier than M_2, in which case M_2 is allowed to fire these actions even if it needs to fire some receive actions first. These receive actions are accumulated in an input context and are expected to be subsequently matched by M_1. Due to the presence of such an input context, the states reached by M_2 during the computation are represented as input trees. The definition first differentiates the type of state p:

Final. In (1), if M_1 is in a final state, then M_2 is in a final state with an empty input context.

Receiving. Case (2) says that if M_1 is in a receiving state, then either (2a) the input context is empty ($T = q$) and M_1 must be able to receive all messages that M_2 can receive; or, (2b) M_1 must be able to consume all the messages at the root of the input tree.

Sending. Case (3) says that if M_1 is a sending state then either: (3a) the input context is empty ($T = q$) and M_2 must be able to send all messages that M_1 can send; or, (3b) M_2 must be able to send every a that M_1 can send, possibly after some receive actions recorded in each $\mathcal{A}_i[q_{i,h}]^{h \in H_i}$. Note that whichever receiving path M_1 chooses in the continuation, M_2 must be able to simulate it, hence the send action $!a$ should be available at the end of each receiving path. Moreover, whenever there are accumulated inputs, we require that $\mathsf{cycle}(!, p)$ does *not* hold, guaranteeing that subtyping preserves orphan-message freedom, i.e., such accumulated receive actions will be eventually executed.

Observe that Case (2) enforces a form of contra-variance for receive actions, while Case (3) enforces a form of covariance for send actions. In Figure 1, we have $M_R \preccurlyeq M_C$ (see § 3).

3 A Sound Algorithm for Asynchronous Subtyping

Our subtyping algorithm takes two machines M_1 and M_2 and produces three possible outputs: *true*, *false*, or *unknown*, which respectively indicate that $M_1 \preccurlyeq M_2$, $M_1 \not\preccurlyeq M_2$, or that the algorithm was unable to prove one of these two results. The algorithm is in three stages. (1) It builds the *simulation tree* of M_1 and M_2 (see Definition 7) that represents sequences of checks between M_1 and M_2, corresponding to the checks in the definition of asynchronous subtyping. Simulation trees may be infinite, but the function terminates whenever: it reaches a node that cannot be expanded, it visits a node whose label has been seen along the path from the root, or it expands a node whose ancestors validate a termination condition that we formalise in Theorem 13. The resulting tree satisfies one of the following conditions: (i) it contains a leaf that could not be expanded because the node represents an unsuccessful check between M_1 and M_2 (in which case the algorithm returns *false*), (ii) all leaves are successful final configurations, see Condition (1) of Definition 5, in which case the algorithm replies *true*, or (iii) for each leaf n it is possible to identify a corresponding ancestor $\mathsf{anc}(n)$. In this last case the tree and the identified ancestors are passed onto the next stage. (2) The algorithm divides the finite tree into several subtrees rooted at those ancestors that do not have other ancestors above them (see the strategy that we outline on page 10). (3) The final stage analyses whether each subtree is of one of the two following kinds. (i) All the leaves in the subtree have the same label as their ancestors: in this case the subtree contains all the needed subtyping checks. (ii) The subtree is a *witness subtree* (see Definition 20), meaning that all the checks that may be considered in continuations of the finite subtree are guaranteed to be successful as well. If all the identified subtrees are of one of the these two kinds, the algorithm replies *true*. Otherwise, it replies *unknown*.

3.1 Generating Asynchronous Simulation Trees

We first define labelled trees, of which our simulation trees are instances; then, we give the operational rules for generating a simulation tree from a pair of communicating machines.

▶ **Definition 6** (Labelled Tree). *A labelled tree is a tree*[2] $(N, n_0, \hookrightarrow, \mathcal{L}, \Sigma, \Gamma)$*, consisting of nodes* N*, root* $n_0 \in N$*, edges* $\hookrightarrow \subseteq N \times \Sigma \times N$*, and node labelling function* $\mathcal{L} : N \longmapsto \Gamma$*.*

Hereafter, we write $n \overset{\sigma}{\hookrightarrow} n'$ when $(n, \sigma, n') \in \hookrightarrow$ and write $n_1 \xhookrightarrow{\sigma_1 \cdots \sigma_k} n_{k+1}$ when there are $n_1, \ldots, n_{k+1}$, such that $n_i \overset{\sigma_i}{\hookrightarrow} n_{i+1}$ for all $1 \leq i \leq k$. We write $n \hookrightarrow n'$ when $n \overset{\sigma}{\hookrightarrow} n'$ for some σ and the label is not relevant. As usual, we write $\hookrightarrow^*$ for the reflexive and transitive closure of $\hookrightarrow$, and $\hookrightarrow^+$ for its transitive closure. Given an edge label $\sigma \in \Sigma$ and two node labels $\alpha, \beta \in \Gamma$, we use $\alpha \overset{\sigma}{\hookrightarrow} \beta$ as a shorthand for $\forall n.\ \mathcal{L}(n) = \alpha \Rightarrow \exists n'.\ \mathcal{L}(n') = \beta \wedge n \overset{\sigma}{\hookrightarrow} n'$. Moreover, we reason up-to tree isomorphism, i.e., two labelled trees are equivalent if there exists a bijective node renaming that preserves both node labelling and labelled transitions.

We can then define simulation trees, labelled trees representing all possible configurations reachable by the asynchronous subtyping simulation game.

▶ **Definition 7** (Simulation Tree). *Let* $M_1 = (P, p_0, \delta_1)$ *and* $M_2 = (Q, q_0, \delta_2)$ *be two communicating machines. Their simulation tree, written* $\mathsf{simtree}(M_1, M_2)$*, is the minimal labelled tree* $(N, n_0, \hookrightarrow, \mathcal{L}, Act, P \times \mathcal{T}_Q)$*, with* $\mathcal{L}(n_0) = p_0 \preccurlyeq q_0$*, generated by the following rules:*

$$\frac{p \xrightarrow{?a} p' \quad q \xrightarrow{?a} q' \quad \mathsf{in}(p) \supseteq \mathsf{in}(q)}{p \preccurlyeq q \ \overset{?a}{\hookrightarrow}\ p' \preccurlyeq q'}\ \textit{(In)} \qquad \frac{p \xrightarrow{!a} p' \quad q \xrightarrow{!a} q' \quad \mathsf{out}(p) \subseteq \mathsf{out}(q)}{p \preccurlyeq q \ \overset{!a}{\hookrightarrow}\ p' \preccurlyeq q'}\ \textit{(Out)}$$

$$\frac{p \xrightarrow{?a_k} p' \quad k \in I \quad \mathsf{in}(p) \supseteq \{a_i \mid i \in I\}}{p \preccurlyeq \langle a_i : \mathcal{A}_i[q_{i,j}]^{j \in J_i} \rangle_{i \in I} \ \overset{?a_k}{\hookrightarrow}\ p' \preccurlyeq \mathcal{A}_k[q_{k,j}]^{j \in J_k}}\ \textit{(InCtx)}$$

$$\frac{\begin{array}{c} p \xrightarrow{!a} p' \qquad\qquad \neg\mathsf{cycle}(!, p) \\ \forall j \in J.\big(\mathsf{inTree}(q_j) = \mathcal{A}_j[q_{j,h}]^{h \in H_j} \wedge \forall h \in H_j.(\mathsf{out}(p) \subseteq \mathsf{out}(q_{j,h}) \wedge q_{j,h} \xrightarrow{!a} q'_{j,h})\big) \end{array}}{p \preccurlyeq \mathcal{A}[q_j]^{j \in J} \overset{!a}{\hookrightarrow} p' \preccurlyeq \mathcal{A}[\mathcal{A}_j[q'_{j,h}]^{h \in H_j}]^{j \in J}}\ \textit{(OutAcc)}$$

Given machines M_1 and M_2, Definition 7 generates a tree whose nodes are labelled by terms of the form $p \preccurlyeq \mathcal{A}[q_i]^{i \in I}$ where p represents the state of M_1, $\mathcal{A}$ represents the receive actions accumulated by M_2, and each q_i represents the state of machine M_2 after each path of accumulated receive actions from the root of $\mathcal{A}$ to the i^{th} hole. Note that we overload the symbol $\preccurlyeq$ used for asynchronous subtyping (Definition 5), however the actual meaning is always made clear by the context. We comment each rule in detail below.

[2] A tree is a connected directed graph without cycles: $\forall n \in N.\ n_0 \hookrightarrow^* n \wedge \forall n, n' \in N.\ n \hookrightarrow^+ n'.\ n \neq n'$.

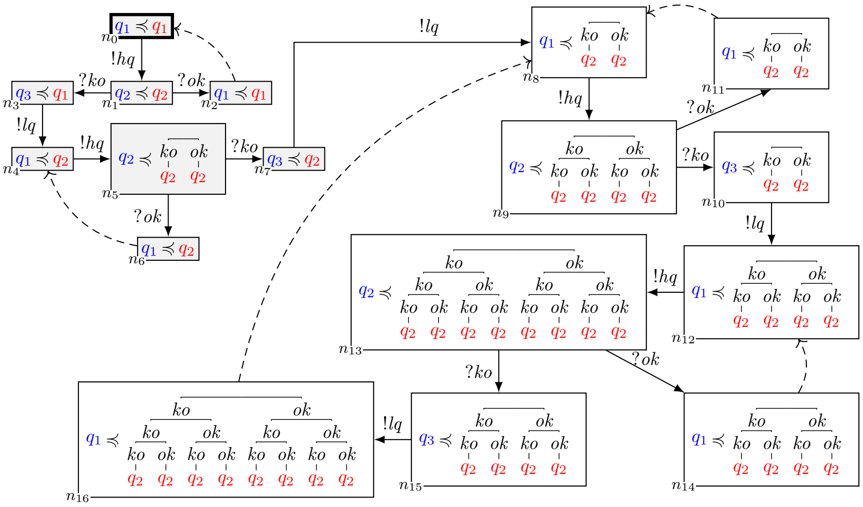


Figure 2 Part of the simulation tree (solid edges only) and candidate tree for $M_R \preccurlyeq M_C$ (Figure 1). The root is circled in thicker line. The node identities are shown at the bottom left of each label.

Rules (In) and (Out) enforce contra-variance of inputs and covariance of outputs, respectively, when no accumulated receive actions are recorded, i.e., $\mathcal{A}$ is a single hole. Rule (In) corresponds to Case (2a) of Definition 5, while rule (Out) corresponds to Case (3a).
Rule (InCtx) is applicable when the input tree $\mathcal{A}$ is non-empty and the state p (of M_1) is able to perform a receive action corresponding to any message located at the root of the input tree (contra-variance of receive actions). This rule corresponds to Case (2b) of Definition 5.
Rule (OutAcc) allows M_2 to execute some receive actions before matching a send action executed by M_1. This rule corresponds to Case (3b) of Definition 5. Intuitively, each send action outgoing from state p must also be eventually executable from each of the states q_j (in M_2) which occur in the input tree $\mathcal{A}[q_j]^{j\in J}$. The possible combinations of receive actions executable from each q_j before executing $!a$ is recorded in $\mathcal{A}_j$, using $\mathsf{inTree}(q_j)$. We assume that the premises of this rule only hold when all invocations of $\mathsf{inTree}(\cdot)$ are defined. Each tree of accumulated receive actions is appended to its respective branch of the input context $\mathcal{A}$, using the notation $\mathcal{A}[\mathcal{A}_j[q'_{j,h}]^{h\in H_j}]^{j\in J}$. The premise $\mathsf{out}(p) \subseteq \mathsf{out}(q_{j,h}) \wedge q_{j,h} \xrightarrow{!a} q'_{j,h}$ guarantees that each $q_{j,h}$ can perform the send actions available from p (covariance of send actions). The additional premise $\neg\mathsf{cycle}(!, p)$ corresponds to that of Case (3b) of Definition 5.

▶ **Example 8.** Figure 2 gives a graphical view of the initial part of the simulation tree $\mathsf{simtree}(M_R, M_C)$. Consider the solid edges only for now. Observe that all branches of the simulation tree are infinite; some traverse nodes with infinitely many different labels, due to the unbounded growth of the input trees (e.g., the one repeatedly performing transitions $!hq\cdot?ko\cdot!lq$); while others traverse nodes with *finitely* many distinct labels (e.g., the one performing first transitions $!hq\cdot?ko\cdot!lq$ and then repeatedly performing $!hq\cdot?ok$).

We adapt the terminology of [20] and say that a node n of $\mathsf{simtree}(M_1, M_2)$ is a *leaf* if it has no successors. A leaf n is *successful* iff $\mathcal{L}(n) = p \preccurlyeq q$, with p and q final; all other leaves are unsuccessful. A *branch* (a full path through the tree) is *successful* iff it is infinite or finishes with a successful leaf; otherwise it is unsuccessful. Using this terminology, we relate asynchronous subtyping (Definition 5) with simulation trees (Definition 7) in Theorem 9.

▶ **Theorem 9.** *Let $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$ be two communicating machines. All branches in* $\mathsf{simtree}(M_1, M_2)$ *are successful if and only if* $M_1 \preccurlyeq M_2$.

3.2 A Simulation Tree-Based Algorithm

Checking whether all branches in $\mathsf{simtree}(M_1, M_2)$ are successful is undecidable. This is a consequence of the undecidability of asynchronous session subtyping [7, 8, 24]. The problem follows from the presence of infinite branches that cannot be algorithmically identified. Our approach is to characterise finite subtrees (called *witness subtrees*) such that all the branches that traverse such finite subtrees are guaranteed to be infinite.

The presentation of our algorithm is in three parts. In Part (1), we give the definition of the kind of *finite* subtree (of a simulation tree) we are interested in (called *candidate* subtrees). In Part (2), we give an algorithm to extract *candidate* subtrees from a simulation tree $\mathsf{simtree}(M_1, M_2)$. In Part (3) we show how to check whether a candidate subtree (which is finite) is a *witness* of infinite branches (hence successful) in the simulation tree.

Part 1. Characterising finite and candidate sub-trees. We define the candidate subtrees of a simulation tree, which are finite subtrees accompanied by an ancestor function mapping each boundary node n to a node located on the path from the root of the tree to n.

▶ **Definition 10** (Finite Subtree). *A finite subtree (r, B) of a labelled tree $S = (N, n_0, \hookrightarrow, \mathcal{L}, \Sigma, \Gamma)$, with r being the subtree root and B the finite set of its leaves (boundary nodes), is the subgraph of S such that: **(1)** $\forall n \in B.\ r \hookrightarrow^* n$; **(2)** $\forall n \in B.\ \nexists n' \in B.\ n \hookrightarrow^+ n'$; and **(3)** $\forall n \in N.\ r \hookrightarrow^* n \implies \exists n' \in B.\ n \hookrightarrow^* n' \vee n' \hookrightarrow^* n$. We use $\mathsf{nodes}(S, r, B) = \{n \in N \mid \exists n' \in B.\ r \hookrightarrow^* n \hookrightarrow^* n'\}$ to denote the (finite) set of nodes of the finite subtree (r, B). Notice that $r \in \mathsf{nodes}(S, r, B)$ and $B \subseteq \mathsf{nodes}(S, r, B)$.*

Condition **(1)** requires that each boundary node can be reached from the root of the subtree. Condition **(2)** guarantees that the boundary nodes are not connected, i.e., they are on different paths from the root. Condition **(3)** enforces that each branch of the tree passing through the root r contains a boundary node.

▶ **Definition 11** (Candidate Subtree). *Let $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$ be two communicating machines with $\mathsf{simtree}(M_1, M_2) = (N, n_0, \hookrightarrow, \mathcal{L}, Act, P \times \mathcal{T}_Q)$. A* candidate subtree *of $\mathsf{simtree}(M_1, M_2)$ is a finite subtree (r, B) paired with a function $\mathsf{anc} : B \longmapsto \mathsf{nodes}(\mathsf{simtree}(M_1, M_2), r, B) \setminus B$ such that, for all $n \in B$, we have: $\mathsf{anc}(n) \hookrightarrow^+ n$ and there are $p, \mathcal{A}, \mathcal{A}', I, J, \{q_j \mid j \in J\}$ and $\{q_i \mid i \in I\}$ such that*

$$\mathcal{L}(n) = p \preccurlyeq \mathcal{A}[q_i]^{i \in I} \ \wedge\ \mathcal{L}(\mathsf{anc}(n)) = p \preccurlyeq \mathcal{A}'[q_j]^{j \in J} \ \wedge\ \{q_i \mid i \in I\} \subseteq \{q_j \mid j \in J\}$$

A candidate subtree is a finite subtree accompanied by a total function on its boundary nodes. The purpose of function anc is to map each boundary node n to a "similar" ancestor n' such that: n' is a node (different from n) on the path from the root r to n (recall that we have $r \notin B$) such that the two labels of n' and n share the same state p of M_1, and the states of M_2 (that populate the holes in the leaves of the input context of the boundary node) are a subset of those considered for the ancestor. We write $img(\mathsf{anc})$ for $\{n \mid \exists n' \in B.\ \mathsf{anc}(n') = n\}$, i.e., $img(\mathsf{anc})$ is the set of ancestors of a given candidate subtree.

▶ **Example 12.** Figure 2 depicts a finite subtree of $\mathsf{simtree}(M_R, M_C)$. The anc function is represented by the dashed edges from boundary nodes to ancestors. We can distinguish distinct candidate subtrees in Figure 2, for instance one rooted at n_0 and with boundary $\{n_2, n_6, n_{11}, n_{14}, n_{16}\}$, another one rooted at n_8 and with boundary $\{n_{11}, n_{14}, n_{16}\}$.

Part 2. Identifying candidate subtrees. We now describe how to generate a finite subtree of the simulation tree, from which we extract candidate subtrees. Since simulation trees are potentially infinite, we need to identify termination conditions (i.e., conditions on nodes that become the boundary of the generated finite subtree).

We first need to define the auxiliary function $\mathsf{extract}(\mathcal{A}, \omega)$, which checks the presence of a sequence of messages ω in an input context $\mathcal{A}$, and extracts the residual input context.

$$\mathsf{extract}(\mathcal{A}, \omega) = \begin{cases} \mathcal{A} & \text{if } \omega = \epsilon \\ \mathsf{extract}(\mathcal{A}_i, \omega') & \text{if } \omega = a_i \cdot \omega', \mathcal{A} = \langle a_j : \mathcal{A}_j \rangle_{j \in J}, \text{ and } i \in J \\ \bot & \text{otherwise} \end{cases}$$

Our termination condition is formalised in Theorem 13 below. This result follows from an argument based on the finiteness of the states of M_1 and of the sets of states from M_2 (which populate the holes of the input contexts in the labels of the nodes in the simulation tree). We write $\mathsf{minHeight}(\mathcal{A})$ for the smallest $\mathsf{height}_i(\mathcal{A})$, with $i \in I(\mathcal{A})$, where $\mathsf{height}_i(\mathcal{A})$ is the length of the path from the root of the input context $\mathcal{A}$ to the i^{th} hole.

▶ **Theorem 13.** *Let $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$ be two communicating machines with $\mathsf{simtree}(M_1, M_2) = (N, n_0, \hookrightarrow, \mathcal{L}, Act, P \times \mathcal{T}_Q)$. For each infinite path $n_0 \hookrightarrow n_1 \hookrightarrow n_2 \cdots \hookrightarrow n_i \hookrightarrow \cdots$ there exist $i < j < k$, with*

$$\mathcal{L}(n_i) = p \preccurlyeq \mathcal{A}_i[q_h]^{h \in H_i} \qquad \mathcal{L}(n_j) = p \preccurlyeq \mathcal{A}_j[q'_h]^{h \in H_j} \qquad \mathcal{L}(n_k) = p \preccurlyeq \mathcal{A}_k[q''_h]^{h \in H_k}$$

s.t. $\{q'_h \mid h \in H_j\} \subseteq \{q_h \mid h \in H_i\}$ and $\{q''_h \mid h \in H_k\} \subseteq \{q_h \mid h \in H_i\}$; and, for $n_i \overset{\psi}{\hookrightarrow} n_j$:

i) $\mathsf{rcv}(\psi) = \omega_1 \cdot \omega_2$ *with ω_1 s.t. $\exists t, z.\ \mathsf{extract}(\mathcal{A}_i, \omega_1) = [\,]_t \wedge \mathsf{extract}(\mathcal{A}_k, \omega_1) = [\,]_z$, or*

ii) $\mathsf{minHeight}(\mathsf{extract}(\mathcal{A}_i, \mathsf{rcv}(\psi))) \leq \mathsf{minHeight}(\mathsf{extract}(\mathcal{A}_k, \mathsf{rcv}(\psi)))$.

Intuitively, the theorem above says that for each infinite branch in the simulation tree, we can find special nodes n_i, n_j and n_k such that the set of states in $\mathcal{A}_j$ (resp. $\mathcal{A}_k$) is included in that of $\mathcal{A}_i$ and the receive actions in the path from n_i to n_j are such that: either (i) only a precise prefix of such actions will be taken from the receive actions accumulated in n_i and n_k or (ii) all of them will be taken from the receive actions in which case n_k must have accumulated more receive actions than n_i. Case (i) deals with infinite branches with only finite labels (hence finite accumulation) while case (ii) considers those cases in which there is unbounded accumulation along the infinite branch.

Based on Theorem 13, the following *algorithm* generates a finite subtree of $\mathsf{simtree}(M_1, M_2)$:

> Compute, initially starting from the root, the branches[3] of $\mathsf{simtree}(M_1, M_2)$ stopping when one of the following types of node is encountered: a leaf, or a node n with a label already seen along the path from the root to n, or a node n_k (with the corresponding node n_i) as those described by the above Theorem 13.

▶ **Example 14.** Consider the finite subtree in Figure 2. It is precisely the finite subtree identified as described above: we stop generating the simulation tree at nodes n_2, n_6, n_{11}, and n_{14} (because their labels have been already seen at the corresponding ancestors n_0, n_4, n_8, and n_{12}) and n_{16} (because of the ancestors n_8 and n_{12} such that n_8, n_{12} and n_{16} correspond to the nodes n_i, n_j and n_k of Theorem 13).

[3] The order nodes are generated is not important (our implementation uses a DFS approach, cf. §4).

When the computed finite subtree contains an unsuccessful leaf, we can immediately conclude that the considered communicating machines are not related. Otherwise, we extract smaller finite subtrees (from the subtree) that are potential candidates to be subsequently checked.

> We define the anc function as follows: for boundary nodes n with an ancestor n' such that $\mathcal{L}(n) = \mathcal{L}(n')$ we define $\mathsf{anc}(n) = n'$; for boundary nodes n_k (with the corresponding node n_i) as those described by in Theorem 13 we define $\mathsf{anc}(n_k) = n_i$. The extraction of the finite subtrees is done by characterising their roots (and taking as boundary the reachable boundary nodes): let $P = \{n \in \mathit{img}(\mathsf{anc}) \mid \exists n'.\ \mathsf{anc}(n') = n \wedge \mathcal{L}(n) \neq \mathcal{L}(n')\}$, the set of such roots is $R = \{n \in P \mid \nexists n' \in P.\ n' \hookrightarrow^{+} n\}$.

Intuitively, to extract subtrees, we restrict our attention to the set P of ancestors with a label different from their corresponding boundary node (corresponding to branches that can generate unbounded accumulation). We then consider the forest of subtrees rooted in nodes in P without an ancestor in P. Notice that for successful leaves we do not define anc; hence, only extracted subtrees without successful nodes have a completely defined anc function. These are candidate subtrees that will be checked as described in the next step.

▶ **Example 15.** Consider the finite subtree in Figure 2. Following the strategy above we extract from it the candidate subtree rooted at n_8 (white nodes), with boundary $\{n_{11}, n_{14}, n_{16}\}$. Note that each ancestor node above n_8 has a label identical to its boundary node.

Part 3. Checking whether the candidate subtrees are witnesses of infinite branches. The final step of our algorithm consists in verifying a property on the identified candidate subtrees which guarantees that all branches traversing the root of the candidate subtree are infinite, hence successful. A candidate subtree satisfies this property when it is also a *witness subtree*, which is the key notion (Definition 20) presented in this third part.

In order for a subtree to be a witness, we require that any behaviour in the simulation tree going beyond the subtree is the infinite repetition of the behaviour already observed in the considered finite subtree. This infinite repetition is only possible if whatever receive actions are accumulated in the input context $\mathcal{A}$ (using Rule (OutAcc)) are eventually executed by the candidate subtype M_1 in Rule (InCtx). The compatibility check between what receive actions can be accumulated and what is eventually executed is done by first synthesising a pair of intermediate automata from a candidate subtree, that represent the possible (repeated) accumulation of the candidate supertype M_2 and the possible (repeated) receive actions of the candidate subtype M_1, and then by checking that these automata are *compatible*. For convenience, we define these intermediate automata as a system of (possibly) mutually recursive equations, which we call a *system of input tree equations*.

▶ **Definition 16** (Input Tree Equations). *Given a set of variables $\mathcal{V}$, ranged over by X, an input tree definition is a term of the grammar $E \quad ::= \quad X \mid \langle a_i : E_i \rangle_{i \in I} \mid \langle E_i \rangle_{i \in I}$.*

A system of input tree equations is a tuple $\mathcal{G} = (\mathcal{V}, X_0, \mathbf{E})$ consisting of a set of variables $\mathcal{V}$, an initial variable $X_0 \in \mathcal{V}$, and with $\mathbf{E}$ consisting of exactly one input tree definition $X \overset{def}{=} E$, with $E \in \mathcal{T}_{\mathcal{V}}$, for each $X \in \mathcal{V}$, where $\mathcal{T}_{\mathcal{V}}$ denotes the input tree definitions on variables $\mathcal{V}$.

Given an input tree definition of the form $\langle a_i : E_i \rangle_{i \in I}$ or $\langle E_i \rangle_{i \in I}$, we assume that $I \neq \emptyset$, $\forall i \neq j \in I.\ a_i \neq a_j$, and that the order of the sub-terms is irrelevant. Whenever convenient, we use set-builder notation to construct an input tree definition, e.g., $\langle E_i \mid i \in I \rangle$. In an input tree equation, the construct $\langle a_i : E_i \rangle_{i \in I}$ represents the capability of accumulating (or actually executing) the receive actions on each message a_i then behaving as in E_i. The construct $\langle E_i \rangle_{i \in I}$ represents a *silent choice* between the different capabilities E_i.

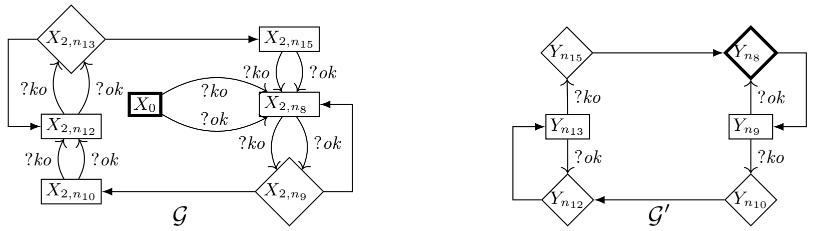


Figure 3 Graphical view of the input tree equations for $M_R \preccurlyeq M_C$ (Figure 1). The starting variables are X_0 and Y_{n_8}. Silent choices are diamond-shaped nodes, other nodes are rectangles.

▶ **Definition 17** (Input Tree Compatibility). *Given two systems of input tree equations $\mathcal{G} = (\mathcal{V}, X_0, \mathbf{E})$ and $\mathcal{G}' = (\mathcal{V}', X'_0, \mathbf{E}')$, such that $\mathcal{V} \cap \mathcal{V}' = \emptyset$, we say that $\mathcal{G}$ is* compatible *with $\mathcal{G}'$, written $\mathcal{G} \sqsubseteq \mathcal{G}'$, if there exists a relation $\mathcal{R} \subseteq \mathcal{T}_{\mathcal{V}} \times \mathcal{T}'_{\mathcal{V}}$ s.t. $(X_0, X'_0) \in \mathcal{R}$ and:*

- *if $(X, E) \in \mathcal{R}$ then $(E', E) \in \mathcal{R}$ with $X \overset{def}{=} E'$;*
- *if $(E, X) \in \mathcal{R}$ then $(E, E') \in \mathcal{R}$ with $X \overset{def}{=} E'$;*
- *if $(\langle E_i \rangle_{i \in I}, E) \in \mathcal{R}$ then $\forall i \in I.\ (E_i, E) \in \mathcal{R}$;*
- *if $(E, \langle E_i \rangle_{i \in I}) \in \mathcal{R}$ then $\forall i \in I.\ (E, E_i) \in \mathcal{R}$;*
- *if $(\langle a_i : E_i \rangle_{i \in I}, \langle a_j : E'_j \rangle_{j \in J}) \in \mathcal{R}$ then $I \subseteq J$ and $\forall i \in I.\ (E_i, E'_i) \in \mathcal{R}$.*

Intuitively, $\mathcal{G} \sqsubseteq \mathcal{G}'$ verifies the compatibility between $\mathcal{G}$, which represents the receive actions that can be accumulated in the context $\mathcal{A}$, and $\mathcal{G}'$, which represents the receive actions that can be actually executed. The first two items of Definition 17 let a variable be replaced by its definition. The next two items explore all the successors of silent choices. The last item guarantees that all the receive actions accumulated in $\mathcal{G}$, cf. Rule (OutAcc), can be actually matched by receive actions in $\mathcal{G}'$, cf. Rule (InCtx).

▶ **Example 18.** A graphical representations of two systems of input tree equations is in Figure 3. We have $\mathcal{G} \sqsubseteq \mathcal{G}'$ since all non-silent choices have the same outgoing transitions.

Before giving the definition of witness subtree, we introduce a few auxiliary functions on which it relies. Given $\omega \in \mathbb{A}^*$, and a state $q \in Q$, we define $\mathsf{accTree}(q, \omega)$ as follows:

$$\mathsf{accTree}(q, \omega) = \begin{cases} [q]_k \text{ with } k \text{ fresh} & \text{if } \omega = \epsilon \\ \mathcal{A}[\mathsf{accTree}(q'_i, \omega')]^{i \in I} & \text{if } \omega = a \cdot \omega', \mathcal{A}[q_i]^{i \in I} = \mathsf{inTree}(q), \forall i \in I.\ q_i \xrightarrow{!a} q'_i \\ \bot & \text{otherwise} \end{cases}$$

Function $\mathsf{accTree}(q, \omega)$ is a key ingredient of the witness subtree definition as it allows for the construction of the accumulation of receive actions (represented as an input tree) that is generated from a state q mimicking the sequence of send actions sending the messages in ω.

We use the auxiliary function $\mathsf{minAcc}(n, q, \psi)$ below to ensure that the effect of performing the transitions from an ancestor to a boundary node is that of increasing (possibly non-strictly) the accumulated receive actions. Here, n represents a known lower bound for the length of a sequence of receive actions accumulated in an input context $\mathcal{A}$, i.e., the length of a path from the root of $\mathcal{A}$ to one of its holes. Assuming that this hole contains the state q, the function returns a lower bound for the length of such a sequence of accumulated receive actions after the transitions in ψ have been executed. Formally, given a natural number n, a sequence of action $\psi \in Act^*$, and a state $q \in Q$ we define this function as follows:

$$\mathsf{minAcc}(n,q,\psi)=\begin{cases} n & \text{if } \psi=\epsilon \\ \mathsf{minAcc}(n-1,q,\psi') & \text{if } \psi=?a\cdot\psi'\wedge n>0 \\ \min_{i\in I}\mathsf{minAcc}(n+\mathsf{height}_i(\mathcal{A}),q_i,\psi') & \text{if } \psi=!a\cdot\psi'\wedge\mathsf{accTree}(q,a)=\mathcal{A}[q_i]^{i\in I} \\ \bot & \text{otherwise} \end{cases}$$

▶ **Example 19.** Consider the transitions from node n_7 to n_9 in Figure 2. There are two send actions $!lq$ and $!hq$ that cannot be directly fired from state q_2 which is a receiving state; the effect is to accumulate receive actions. Such an accumulation is computed by $\mathsf{accTree}(q_2, lq\cdot hq) = \langle ko : \langle ko : q_2, ok : q_2\rangle, ok : \langle ko : q_2, ok : q_2\rangle\rangle$. For this sequence of transitions, the effect on the (minimal) length of the accumulated receive actions can be computed by $\mathsf{minAcc}(0, q_2, !lq\cdot!hq) = 2$; meaning that before executing the sequence of transitions $!lq\cdot!hq$ state q_2 has not accumulated receive actions in front, while at the end an input context with minimal depth 2 is generated as accumulation.

We finally give the definition of witness subtree.

▶ **Definition 20** (Witness Subtree). *Let $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$ be two communicating machines with $\mathsf{simtree}(M_1, M_2) = (N, n_0, \hookrightarrow, \mathcal{L}, Act, P \times \mathcal{T}_Q)$. A candidate subtree of $\mathsf{simtree}(M_1, M_2)$ with root r and boundary B is a* witness *if the following holds:*

1. *For all $n \in B$, given ψ such that $\mathsf{anc}(n) \overset{\psi}{\hookrightarrow} n$, we have $|\mathsf{rcv}(\psi)| > 0$.*
2. *For all $n \in img(\mathsf{anc})$ and $n' \in img(\mathsf{anc}) \cup B$ such that $n \overset{\psi}{\hookrightarrow} n'$, $\mathcal{L}(n) = p \preccurlyeq \mathcal{A}[q_i]^{i\in I}$, and $\mathcal{L}(n') = p' \preccurlyeq \mathcal{A}'[q_j]^{j\in J}$, we have that $\forall i \in I$:*
   - **a.** *$\{q_h \mid h \in H \text{ s.t. } \mathsf{accTree}(q_i, \mathsf{snd}(\psi)) = \mathcal{A}''[q_h]^{h\in H}\} \subseteq \{q_j \mid j \in J\}$;*
   - **b.** *if $n' \in B$ then $\mathsf{minAcc}(\mathsf{minHeight}(\mathcal{A}), q_i, \psi) \geq \mathsf{minHeight}(\mathcal{A})$.*
3. *$\mathcal{G} \sqsubseteq \mathcal{G}'$ where*
   - **a.** *$\mathcal{G} = (\{X_0\} \cup \{X_{q,n} \mid q \in Q, n \in \mathsf{nodes}(S, r, B)\backslash B\}, X_0, \mathbf{E})$ with $\mathbf{E}$ defined as follows:*
     - **i.** *$X_0 \overset{def}{=} T\{X_{q,r}/q \mid q \in Q\}$, with $\mathcal{L}(r) = p \preccurlyeq T$*
     - **ii.** *$X_{q,n} \overset{def}{=}$*

$$\begin{cases} \langle X_{q,tr(n')} \mid \exists a.n \overset{?a}{\hookrightarrow} n'\rangle & \textit{if } \exists a.n \overset{?a}{\hookrightarrow} \\ \langle \mathcal{A}[X_{q'_i,tr(n')}]^{i\in I} \mid \exists a.n \overset{!a}{\hookrightarrow} n' \wedge \mathsf{inTree}(q) = \mathcal{A}[q_i]^{i\in I} \wedge \forall i \in I.q_i \xrightarrow{!a} q'_i\rangle & \textit{otherwise} \end{cases}$$

   - **b.** *$\mathcal{G}' = (\{Y_n \mid n \in \mathsf{nodes}(S, r, B)\backslash B\}, Y_r, \mathbf{E}')$ with $\mathbf{E}'$ defined as follows:*

$$Y_n \overset{def}{=} \begin{cases} \langle Y_{tr(n')} \mid n \overset{!a}{\hookrightarrow} n'\rangle & \textit{if } \exists n'.n \overset{!a}{\hookrightarrow} n' \\ \langle a : Y_{tr(n')} \mid n \overset{?a}{\hookrightarrow} n'\rangle & \textit{if } \exists n'.n \overset{?a}{\hookrightarrow} n' \end{cases} \qquad \textit{with } tr(n) = n, \textit{ if } n \notin B;\ tr(n) = \mathsf{anc}(n), \textit{ otherwise.}$$

Condition (1) requires the existence of a receive transition between an ancestor and a boundary node. This implies that if the behaviour beyond the witness subtree is the repetition of behaviour already observed in the subtree, then there cannot be send-only cycles. Condition (2a) requires that the transitions from ancestors to boundary nodes (or to other ancestors) are such that they include those behaviour that can be computed by the $\mathsf{accTree}$ function. We assume that this condition does not hold if $\mathsf{accTree}(q_i, \mathsf{snd}(\psi)) = \bot$ for any $i \in I$; hence the states q_i of M_2 in an ancestor are able to mimic all the send actions performed by M_1 along the sequences of transitions in the witness subtree starting from the considered ancestor. Condition (2b) ensures that by repeating transitions from ancestors to boundary nodes, the accumulation of receive actions is, overall, increasing. In other words, the rate at which accumulation is taking place is higher than the rate at which the context is reduced by Rule (InCtx). Condition (3) checks that the receive actions that can be accumulated by M_2(represented by $\mathcal{G}$) and those that are expected to be actually executed by M_1 (represented by $\mathcal{G}'$) are compatible. In $\mathcal{G}$, there is an equation for the root node and for each pair consisting of a local state in M_2 and a node n in the witness subtree. The equation for the root node is given in (3(a)i), where we simply transform an input context into an input tree definition. The other equations are given in (3(a)ii), where we use the partial function $\mathsf{inTree}(q)$. Each equation represents what can be accumulated by starting from node n (focusing on local state q). In $\mathcal{G}'$, there is an equation for each node n in the witness subtree, as defined in (3b) There are two types of equations depending on type of transitions outgoing from node n. A send transition leads to silent choices, while receive transitions generate corresponding receive choices.

▶ **Example 21.** The candidate subtree rooted at n_8 in Figure 2 satisfies Definition 2. (1) Each path from an ancestor to a boundary node includes at least one receive action. (2a) For each sequence of transitions from an ancestor to a boundary node (or another ancestor) the behaviour of the states of M_2, as computed by the $\mathsf{accTree}$ function, has already been observed. (2b) For each sequence of transitions from an ancestor to a boundary node, the rate at which receive actions are accumulated is higher than or equal to the rate at which they are removed from the accumulation. (3) The systems of input tree equations $\mathcal{G}$ (3a) and $\mathcal{G}'$ (3b) are given in Figure 3, and are compatible, see Example 18.

We conclude by stating our main result; given a simulation tree with a witness subtree with root r, all the branches in the simulation tree traversing r are infinite (hence successful).

▶ **Theorem 22.** *Let $M_1 = (P, p_0, \delta_1)$ and $M_2 = (Q, q_0, \delta_2)$ be two communicating machines with $\mathsf{simtree}(M_1, M_2) = (N, n_0, \hookrightarrow, \mathcal{L}, Act, P \times \mathcal{T}_Q)$. If $\mathsf{simtree}(M_1, M_2)$ has a witness subtree with root r then for every node $n \in N$ such that $r \hookrightarrow^* n$ there exists n' such that $n \hookrightarrow n'$.*

Hence, we can conclude that if the candidate subtrees of $\mathsf{simtree}(M_1, M_2)$ identified following the strategy explained in Part (2) are also witness subtrees, then we have $M_1 \preccurlyeq M_2$.

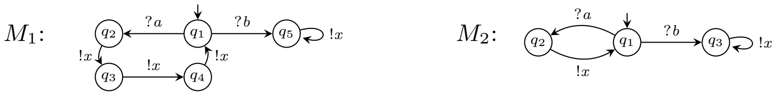
▶ Remark 23. When our algorithm finds a successful leaf, a previously seen label, or a witness subtree in each branch then the machines are in the subtyping relation. If an unsuccessful leaf is found (while generating the initial finite subtree as described in Part (2)), then the machines are *not* in the subtyping relation. In all other cases, the algorithm is unable to give a decisive verdict (i.e., the result is *unknown*). There are two possible causes for an unknown result: either (*i*) it is impossible to extract a forest of candidate subtrees (i.e., there are successful leaves below some ancestor) or (*ii*) some candidate subtree is not a witness.

4 Evaluation, Related Work, and Conclusions

Evaluation. To evaluate the cost and applicability of our algorithm, we have produced a faithful implementation of it, which constructs the simulation tree in a depth-first search manner, while recording the nodes visited in different branches to avoid re-computing several times the same subtrees. We have run our tool on 174 tests which were either taken from the literature on asynchronous subtyping [10, 24], or handcrafted to test the limits of our approach. All of these tests terminate under a second. Out of these tests, 92 are *negative* (the types are not in the subtyping relation) and our tool gives the expected result (*false*) for all of them. The other 82 tests are *positive* (the types are in the subtyping relation) and our tool gives the expected result (*true*) for all but 8 tests, for which it returns *unknown*. All of these 8 examples feature complex accumulation patterns that our theory cannot recognise, e.g., Example 24. The implementation and our test data are available on our GitHub repository [5].

The implementation includes an additional optimisation which performs a check for $\overline{M_2} \preccurlyeq \overline{M_1}$ (relying on a previous result showing that $M_1 \preccurlyeq M_2 \iff \overline{M_2} \preccurlyeq \overline{M_1}$ [7,24]) when the result of checking $M_1 \preccurlyeq M_2$ is *unknown*.

▶ **Example 24.** Given the machines below, $\mathsf{simtree}(M_1, M_2)$ contains infinitely many nodes with labels of the form: $q_1 \preccurlyeq \langle a : \langle a : \langle a : \langle \cdots \rangle, b : q_3 \rangle, b : q_3 \rangle, b : q_3 \rangle$.



Each of these nodes has two successors, one where $?a$ is fired (the machines stay in the larger loop), and one where $?b$ is fired (the machines move to their smaller loop). The machines can always enter this send-only cycle, hence Condition (1) of Definition 20 never applies.

Related work. Gay and Hole [16,17] introduced (synchronous) subtyping for session types and show it is decidable. Mostrous et al. [27] adapted the notion of session subtyping to asynchronous communication, by introducing delayed inputs. Later, Chen et al. [10,11] provided an alternative definition prohibiting orphan messages, we used this definition in this work. Recently, asynchronous subtyping was shown to be undecidable by encoding it as an equivalent question in the setting of Turing machines [24] and queue machines [7]. Recent work [7,8,24] investigated restrictions to achieve decidability, these restrictions are either on the size of the FIFO channels or syntactical. In the latter case, we recall the single-out and single-in restrictions, i.e., where all output (respectively input) choices are singletons.

The relationship between communicating machines and (multiparty asynchronous) session types has been studied in [13,14]. Communicating machines are Turing-complete, hence most of their properties are undecidable [6]. Many variations have been introduced in order to recover decidability, e.g., using (existential or universal) bounds [18], restricting to different types of topologies [22,31], or using bag or lossy channels instead of FIFO queues [1,2,9,12].

Conclusions and future work. We have proposed a sound algorithm for checking asynchronous session subtyping, showing that it is still possible to decide whether two types are related for many nontrivial examples. Our algorithm is based on a (potentially infinite) tree representation of the coinductive definition of asynchronous subtyping; it checks for the presence of finite witnesses of infinite successful subtrees. We have provided an implementation and applied it to examples that cannot be recognised by previous approaches. Although the (worst-case) complexity of our algorithm is rather high (the termination condition expects to encounter a set of states already encountered, of which there may be exponentially many), our implementation shows that it actually terminates under a second for machines of size comparable to typical communication protocols used in real programs, e.g., Go programs feature between three and four communication primitives per channel and whose branching construct feature two branches, on average [15].

As future work, we plan to enrich our algorithm to recognise subtypes featuring more complex accumulation patterns, e.g., Example 24. Moreover, due to the tight correspondence with safety of communicating machines [24], we plan to investigate the possibility of using our approach to characterise a novel decidable subclass of communicating machines.

References

1 Parosh Aziz Abdulla, Ahmed Bouajjani, and Bengt Jonsson. On-the-Fly Analysis of Systems with Unbounded, Lossy FIFO Channels. In *CAV 1998*, pages 305–318, 1998. doi:10.1007/BFb0028754.

2 Parosh Aziz Abdulla and Bengt Jonsson. Verifying Programs with Unreliable Channels. In *(LICS 1993)*, pages 160–170, 1993. doi:10.1109/LICS.1993.287591.

3 Roberto M. Amadio and Luca Cardelli. Subtyping Recursive Types. *ACM Trans. Program. Lang. Syst.*, 15(4):575–631, 1993. doi:10.1145/155183.155231.

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. *Foundations and Trends in Programming Languages*, 3(2-3):95–230, 2016. doi:10.1561/2500000031.

5 The Authors. A sound algorithm for asynchronous session subtyping. https://github.com/julien-lange/asynchronous-subtyping, 2019.

6 Daniel Brand and Pitro Zafiropulo. On Communicating Finite-State Machines. *J. ACM*, 30(2):323–342, 1983. doi:10.1145/322374.322380.

7 Mario Bravetti, Marco Carbone, and Gianluigi Zavattaro. Undecidability of asynchronous session subtyping. *Inf. Comput.*, 256:300–320, 2017.

8 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.

9 Gérard Cécé, Alain Finkel, and S. Purushothaman Iyer. Unreliable Channels are Easier to Verify Than Perfect Channels. *Inf. Comput.*, 124(1):20–31, 1996. doi:10.1006/inco.1996.0003.

10 Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. On the Preciseness of Subtyping in Session Types. *Logical Methods in Computer Science*, 13(2), 2017. doi:10.23638/LMCS-13(2:12)2017.

11 Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. On the Preciseness of Subtyping in Session Types. In *PPDP 2014*, pages 146–135. ACM Press, 2014.

12 Lorenzo Clemente, Frédéric Herbreteau, and Grégoire Sutre. Decidable Topologies for Communicating Automata with FIFO and Bag Channels. In *CONCUR 2014*, pages 281–296, 2014. doi:10.1007/978-3-662-44584-6_20.

13 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty Session Types Meet Communicating Automata. In *ESOP 2012*, pages 194–213, 2012. doi:10.1007/978-3-642-28869-2_10.

14 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. In *ICALP 2013*, pages 174–186, 2013. doi:10.1007/978-3-642-39212-2_18.

15 N. Dilley and J. Lange. An Empirical Study of Messaging Passing Concurrency in Go Projects. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 377–387, February 2019. doi:10.1109/SANER.2019.8668036.

16 Simon J. Gay and Malcolm Hole. Types and Subtypes for Client-Server Interactions. In *ESOP 1999*, pages 74–90, 1999. doi:10.1007/3-540-49099-X_6.

17 Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Inf.*, 42(2-3):191–225, 2005. doi:10.1007/s00236-005-0177-z.

18 Blaise Genest, Dietrich Kuske, and Anca Muscholl. On Communicating Automata with Bounded Channels. *Fundam. Inform.*, 80(1-3):147–167, 2007. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi80-1-3-09.

19 Raymond Hu and Nobuko Yoshida. Hybrid Session Verification Through Endpoint API Generation. In *FASE 2016*, pages 401–418, 2016. doi:10.1007/978-3-662-49665-7_24.

20 Petr Jancar and Faron Moller. Techniques for Decidability and Undecidability of Bisimilarity. In *CONCUR 1999*, pages 30–45, 1999. doi:10.1007/3-540-48320-9_5.

21 Dexter Kozen, Jens Palsberg, and Michael I. Schwartzbach. Efficient Recursive Subtyping. *Mathematical Structures in Computer Science*, 5(1):113–125, 1995. doi:10.1017/S0960129500000657.

22 Salvatore La Torre, P. Madhusudan, and Gennaro Parlato. Context-Bounded Analysis of Concurrent Queue Systems. In *TACAS 2008*, pages 299–314, 2008. doi:10.1007/978-3-540-78800-3_21.

23 Julien Lange and Nobuko Yoshida. Characteristic Formulae for Session Types. In *TACAS*, volume 9636 of *Lecture Notes in Computer Science*, pages 833–850. Springer, 2016.

24 Julien Lange and Nobuko Yoshida. On the Undecidability of Asynchronous Session Subtyping. In *FoSSaCS*, volume 10203 of *Lecture Notes in Computer Science*, pages 441–457, 2017.

25 Sam Lindley and J. Garrett Morris. Embedding session types in Haskell. In *Haskell 2016*, pages 133–145, 2016. doi:10.1145/2976002.2976018.

26 Dimitris Mostrous and Nobuko Yoshida. Session typing and asynchronous subtyping for the higher-order π-calculus. *Inf. Comput.*, 241:227–263, 2015. doi:10.1016/j.ic.2015.02.002.

27 Dimitris Mostrous, Nobuko Yoshida, and Kohei Honda. Global Principal Typing in Partially Commutative Asynchronous Sessions. In *ESOP 2009*, pages 316–332, 2009. doi:10.1007/978-3-642-00590-9_23.

28 Rumyana Neykova, Raymond Hu, Nobuko Yoshida, and Fahd Abdeljallal. A Session Type Provider: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F$\sharp$. In *CC 2018*. ACM, 2018.

29 Dominic A. Orchard and Nobuko Yoshida. Effects as sessions, sessions as effects. In *POPL 2016*, pages 568–581, 2016. doi:10.1145/2837614.2837634.

30 Luca Padovani. A simple library implementation of binary sessions. *J. Funct. Program.*, 27:e4, 2017. doi:10.1017/S0956796816000289.

31 Wuxu Peng and S. Purushothaman. Analysis of a Class of Communicating Finite State Machines. *Acta Inf.*, 29(6/7):499–522, 1992. doi:10.1007/BF01185558.

32 Alceste Scalas and Nobuko Yoshida. Lightweight Session Programming in Scala. In *ECOOP 2016*, pages 21:1–21:28, 2016. doi:10.4230/LIPIcs.ECOOP.2016.21.

Domain-Aware Session Types

Luís Caires
Universidade Nova de Lisboa, Portugal

Jorge A. Pérez
University of Groningen, The Netherlands

Frank Pfenning
Carnegie Mellon University, Pittsburgh, PA, USA

Bernardo Toninho
Universidade Nova de Lisboa, Portugal

Abstract

We develop a generalization of existing Curry-Howard interpretations of (binary) session types by relying on an extension of linear logic with features from *hybrid logic*, in particular modal worlds that indicate *domains*. These worlds govern *domain migration*, subject to a parametric accessibility relation familiar from the Kripke semantics of modal logic. The result is an expressive new typed process framework for domain-aware, message-passing concurrency. Its logical foundations ensure that well-typed processes enjoy session fidelity, global progress, and termination. Typing also ensures that processes only communicate with accessible domains and so respect the accessibility relation.

Remarkably, our domain-aware framework can specify scenarios in which domain information is available only at runtime; flexible accessibility relations can be cleanly defined and statically enforced. As a specific application, we introduce domain-aware *multiparty session types*, in which global protocols can express arbitrarily nested sub-protocols via domain migration. We develop a precise analysis of these multiparty protocols by reduction to our binary domain-aware framework: complex domain-aware protocols can be reasoned about at the right level of abstraction, ensuring also the principled transfer of key correctness properties from the binary to the multiparty setting.

2012 ACM Subject Classification Theory of computation → Process calculi; Theory of computation → Type structures; Software and its engineering → Message passing

Keywords and phrases Session Types, Linear Logic, Process Calculi, Hybrid Logic

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.39

Related Version Omitted proofs and extended examples: `http://arxiv.org/abs/1907.01318`.

Funding Caires and Toninho are supported by NOVA LINCS (Ref. UID/CEC/04516/2019). Pérez is supported by the NWO VIDI Project No. 016.Vidi.189.046: "Unifying Correctness for Communicating Software". Pfenning is supported by NSF Grant No. CCF-1718267: "Enriching Session Types for Practical Concurrent Programming".

1 Introduction

The goal of this paper is to show how existing Curry-Howard interpretations of session types [10, 11] can be generalized to a *domain-aware* setting by relying on an extension of linear logic with features from *hybrid logic* [40, 5]. These extended logical foundations of message-passing concurrency allow us to analyze complex domain-aware concurrent systems (including those governed by multiparty protocols) in a precise and principled manner.

Software systems typically rely on *communication* between heterogeneous services; at their heart, these systems rely on message-passing protocols that combine mobility, concurrency, and distribution. As distributed services are often virtualized, protocols should span diverse software and hardware *domains*. These domains can have multiple interpretations, such as the location where services reside, or the principals on whose behalf they act. Concurrent

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 39; pp. 39:1–39:17
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

behavior is then increasingly *domain-aware*: a partner's potential for interaction is influenced not only by the domains it is involved in at various protocol phases (its context), but also by *connectedness* relations among domains. Moreover, domain architectures are rarely fully specified: to aid modularity and platform independence, system participants (e.g., developers, platform vendors, service clients) often have only partial views of actual domain structures. Despite their importance in communication correctness and trustworthiness at large, the formal status of domains within *typed* models of message-passing systems remains unexplored.

This paper contributes to typed approaches to the analysis of domain-aware communications, with a focus on *session-based concurrency.* This approach specifies the intended message-passing protocols as *session types* [30, 31, 24]. Different type theories for *binary* and *multiparty* (n-ary) protocols have been developed. In both cases, typed specifications can be conveniently coupled with π-calculus processes [36], in which so-called session channels connect exactly two subsystems. Communication correctness usually results from two properties: *session fidelity* (type preservation) and *deadlock freedom* (progress). The former says that well-typed processes always evolve to well-typed processes (a safety property); the latter says that well-typed processes will never get into a stuck state (a liveness property).

A key motivation for this paper is the sharp contrast between (a) the growing relevance of domain-awareness in message-passing, concurrent systems and (b) the expressiveness of existing session type frameworks, binary and multiparty, which cannot adequately specify (let alone enforce) domain-related requirements. Indeed, existing session types frameworks, including those based on Curry-Howard interpretations [10, 47, 14], capture communication behavior at a level of abstraction in which even basic domain-aware assertions (e.g., "*Shipper* resides in domain AmazonUS") cannot be expressed. As an unfortunate consequence, the effectiveness of the analysis techniques derived from these frameworks is rather limited.

To better illustrate our point, consider a common distributed design pattern: a middleware agent (mw) which answers requests from clients (cl), sometimes offloading the requests to a server (serv) to better manage local resource availability. In the framework of multiparty session types [32] this protocol can be represented as the global type:

$$\begin{aligned}&\mathtt{cl} \twoheadrightarrow \mathtt{mw}{:}\{\mathit{request}\langle \mathit{req}\rangle.\ \mathtt{mw} \twoheadrightarrow \mathtt{cl}{:}\{\ \mathit{reply}\langle \mathit{ans}\rangle.\ \mathtt{mw} \twoheadrightarrow \mathtt{serv}{:}\{\mathit{done}.\mathtt{end}\}\ ,\ \ \mathit{wait}.\mathtt{mw} \twoheadrightarrow \mathtt{serv}{:}\{\mathit{req}\langle \mathit{data}\rangle.\\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathtt{serv} \twoheadrightarrow \mathtt{mw}{:}\{\mathit{reply}\langle \mathit{ans}\rangle.\mathtt{mw} \twoheadrightarrow \mathtt{cl}{:}\{\mathit{reply}\langle \mathit{ans}\rangle.\mathtt{end}\}\}\}\}\}\end{aligned}$$

The client first sends a request to the middleware, which answers back with either a *reply* message containing the answer or a *wait* message, signaling that the server will be contacted to produce the final *reply.* While this multiparty protocol captures the intended communication behavior, it does not capture that protocols for the middleware and the server often involve some form of privilege escalation or specific authentication – ensuring, e.g., that the server interaction is adequately isolated from the client, or that the escalation must precede the server interactions. These requirements simply cannot be represented in existing frameworks.

Our work addresses this crucial limitation by generalizing Curry-Howard interpretations of session types by appealing to hybrid logic features. We develop a logically motivated typed process framework in which *worlds* from modal logics precisely and uniformly define the notion of *domain* in session-based concurrency. At the level of *binary* sessions, domains manifest themselves through point-to-point domain migration and communication. In *multiparty* sessions, domain migration is specified choreographically through the new construct $\mathsf{p}\ \mathsf{moves}\ \tilde{\mathsf{q}}\ \mathsf{to}\ \omega\ \mathsf{for}\ G_1\ ;\ G_2$, where participant $\mathtt{p}$ leads a migration of participants $\tilde{\mathtt{q}}$ to domain ω in order to perform protocol G_1, who then migrate back to perform protocol G_2.

Consider the global type $\mathit{Offload} \triangleq \mathtt{mw} \twoheadrightarrow \mathtt{serv}{:}\{\mathit{req}\langle \mathit{data}\rangle.\mathtt{serv} \twoheadrightarrow \mathtt{mw}{:}\{\mathit{reply}\langle \mathit{ans}\rangle.\mathtt{end}\}\}$ in our previous example. Our framework allows us to refactor the global type above as:

$$\begin{aligned}&\mathtt{cl} \twoheadrightarrow \mathtt{mw}{:}\{\mathit{request}\langle \mathit{req}\rangle.\ \mathtt{mw} \twoheadrightarrow \mathtt{cl}{:}\{\ \mathit{reply}\langle \mathit{ans}\rangle.\mathtt{mw} \twoheadrightarrow \mathtt{serv}{:}\{\mathit{done}.\mathtt{end}\}\ \ ,\ \ \mathit{wait}.\mathtt{mw} \twoheadrightarrow \mathtt{serv}{:}\{\mathit{init}.\\ &\qquad\qquad\qquad\qquad\qquad \mathtt{mw}\ \mathsf{moves}\ \mathtt{serv}\ \mathsf{to}\ w_{\mathtt{priv}}\ \mathsf{for}\ \mathit{Offload}\ ;\ \mathtt{mw} \twoheadrightarrow \mathtt{cl}{:}\{\mathit{reply}\langle \mathit{ans}\rangle.\mathtt{end}\}\}\}\}\end{aligned}$$

By considering a first-class multiparty domain migration primitive at the type and process levels, we can specify that the *offload* portion of the protocol takes place after the middleware and the server *migrate* to a private domain $w_{\texttt{priv}}$, as well as ensuring that only accessible domains can be interacted with. For instance, the type for the server that is mechanically *projected* from the protocol above ensures that the server first migrates to the private domain, communicates with the middleware, and then migrates back to its initial domain.

Perhaps surprisingly, our domain-aware *multiparty* sessions are studied within a context of logical *binary* domain-aware sessions, arising from a propositions-as-types interpretation of hybrid linear logic [22, 18], with strong static correctness guarantees derived from the logical nature of the system. Multiparty domain-awareness arises through an interpretation of multiparty protocols as *medium processes* [7] that orchestrate the multiparty interaction while enforcing the necessary domain-level constraints and migration steps.

Contributions. The key contributions of this work are:

1. A process model with explicit domain-based migration (§ 2). We present a session π-calculus with domains that can be communicated via novel domain movement prefixes.
2. A session type discipline for domain-aware interacting processes (§ 3). Building upon an extension of linear logic with features from *hybrid logic* [22, 18] we generalize the Curry-Howard interpretation of session types [10, 11] by interpreting *(modal) worlds* as *domains* where session behavior resides. In our system, types can specify domain *migration* and *communication*; domain mobility is governed by a parametric accessibility relation. Judgments stipulate the services used and realized by processes *and* the domains where sessions should be present. Our type discipline statically enforces session fidelity, global progress and, notably, that communication can only happen between accessible domains.
3. As a specific application, we introduce a framework of domain-aware multiparty sessions (§ 4) that uniformly extends the standard multiparty session framework of [32] with domain-aware migration and communication primitives. Our development leverages our logically motivated domain-aware *binary* sessions (§ 3) to give a precise semantics to multiparty sessions through a (typed) *medium process* that acts as an orchestrator of domain-aware multiparty interactions, lifting the strong correctness properties of typed processes to the multiparty setting. We show that mediums soundly and completely encode the local behaviors of participants in a domain-aware multiparty session.

We conclude with a discussion of related work (§ 5) and concluding remarks (§ 6).

2 Process Model

We introduce a synchronous π-calculus [42] with labeled choice and explicit domain migration and communication. We write $\omega, \omega', \omega''$ to stand for a concrete domain $(w, w', \ldots)$ or a domain variable $(\alpha, \alpha', \ldots)$. Domains are handled at a high-level of abstraction, with their identities being attached to session channels. Just as the π-calculus allows for communication over names and name mobility, our model also allows for domain communication and mobility. These features are justified with the typing discipline of § 3.

▶ **Definition 2.1.** *Given infinite, disjoint sets* Λ *of* names (x, y, z, u, v), $\mathcal{L}$ *of labels* $l_1, l_2, \ldots$, $\mathcal{W}$ *of* domain tags (w, w', w'') *and* $\mathcal{V}$ *of* domain variables (α, β, γ), *respectively, the set of* processes (P, Q, R) *is defined by*

$$\begin{array}{rcllllll}
P & ::= & \mathbf{0} & \mid\ P \mid Q & \mid\ (\boldsymbol{\nu} y)P & \mid\ x\langle y\rangle.P & \mid\ x(y).P & \mid\ !x(y).P \\
& \mid & [x \leftrightarrow y] & \mid\ x \triangleright \{l_i : P_i\}_{i \in I} & \mid\ x \triangleleft l_i; P & & & \\
& \mid & x\langle y@\omega\rangle.P & \mid\ x(y@\omega).P & \mid\ x\langle\omega\rangle.P & \mid\ x(\alpha).P & &
\end{array}$$

Domain-aware prefixes are present only in the last line. As we make precise in the typed setting of § 3, these constructs realize mobility and domain communication, in the usual sense of the π-calculus: migration to a domain is always associated to mobility with a fresh name.

The operators $\mathbf{0}$ (inaction), $P \mid Q$ (parallel composition) and $(\boldsymbol{\nu} y)P$ (name restriction) are standard. We then have $x\langle y\rangle.P$ (send y on x and proceed as P), $x(y).P$ (receive z on x and proceed as P with parameter y replaced by z), and $!x(y).P$ which denotes replicated (persistent) input. The forwarding construct $[x \leftrightarrow y]$ equates x and y; it is a primitive representation of a copycat process. The last two constructs in the second line define a labeled choice mechanism: $x \triangleright \{l_i : P_i\}_{i\in I}$ is a process that awaits some label l_j (with $j \in I$) and proceeds as P_j. Dually, the process $x \triangleleft l_i; P$ emits a label l_i and proceeds as P.

The first two operators in the third line define explicit domain migration: given a domain ω, $x\langle y@\omega\rangle.P$ denotes a process that is prepared to migrate the communication actions in P on endpoint x, to session y on ω. Complementarily, process $x(y@\omega).P$ signals an endpoint x to move to ω, providing P with the appropriate session endpoint that is then bound to y. In a typed setting, domain movement will be always associated with a fresh session channel. Alternatively, this form of coordinated migration can be read as an explicit form of agreement (or authentication) in trusted domains. Finally, the last two operators in the third line define output and input of domains, $x\langle\omega\rangle.P$ and $x(\alpha).P$, respectively. These constructs allow for domain information to be obtained and propagated across processes dynamically.

Following [41], we abbreviate $(\boldsymbol{\nu} y)x\langle y\rangle$ and $(\boldsymbol{\nu} y)x\langle y@\omega\rangle$ as $\overline{x}\langle y\rangle$ and $\overline{x}\langle y@\omega\rangle$, respectively. In $(\boldsymbol{\nu} y)P$, $x(y).P$, and $x(y@\omega).P$ the distinguished occurrence of name y is binding with scope P. Similarly for α in $x(\alpha).P$. We identify processes up to consistent renaming of bound names and variables, writing $\equiv_\alpha$ for this congruence. $P\{x/y\}$ denotes the capture-avoiding substitution of x for y in P. While *structural congruence* $\equiv$ expresses standard identities on the basic structure of processes (cf. [9]), *reduction* expresses their behavior.

Reduction ($P \to Q$) is the binary relation defined by the rules below and closed under structural congruence; it specifies the computations that a process performs on its own.

$$\begin{array}{ll}
x\langle y\rangle.Q \mid x(z).P \to Q \mid P\{y/z\} & x\langle y\rangle.Q \mid !x(z).P \to Q \mid P\{y/z\} \mid !x(z).P \\
x\langle y@\omega\rangle.P \mid x(z@\omega').Q \to P \mid Q\{y/z\} & x\langle\omega\rangle.P \mid x(\alpha).Q \to P \mid Q\{\omega/\alpha\} \\
(\boldsymbol{\nu} x)([x \leftrightarrow y] \mid P) \to P\{y/x\} & Q \to Q' \Rightarrow P \mid Q \to P \mid Q' \\
P \to Q \Rightarrow (\boldsymbol{\nu} y)P \to (\boldsymbol{\nu} y)Q & x \triangleleft l_j; P \mid x \triangleright \{l_i : Q_i\}_{i\in I} \to P \mid Q_j \quad (j \in I)
\end{array}$$

For the sake of generality, reduction allows dual endpoints with the same name to interact, independently of the domains of their subjects. The type system introduced next will ensure, among other things, *local reductions*, disallowing synchronisations among distinct domains.

3 Domain-aware Session Types via Hybrid Logic

This section develops a new domain-aware formulation of binary session types. Our system is based on a Curry-Howard interpretation of a linear variant of so-called *hybrid logic*, and can be seen as an extension of the interpretation of [10, 11] to hybrid (linear) logic. Hybrid logic is often used as an umbrella term for a class of logics that extend the expressiveness of propositional logic by considering modal *worlds* as syntactic objects that occur in propositions.

As in [10, 11], propositions are interpreted as session types of communication channels, proofs as typing derivations, and proof reduction as process communication. As main novelties, here we interpret: logical worlds as *domains*; the hybrid connective $@_\omega A$ as the type of a session that *migrates* to an accessible domain ω; and type-level quantification over worlds $\forall\alpha.A$ and $\exists\alpha.A$ as *domain communication*. We also consider a type-level operator

$\downarrow\alpha.A$ (read "here") which binds the *current* domain of the session to α in A. The syntax of domain-aware session types is given in Def. 3.1, where $w, w_1, \ldots$ stand for domains drawn from $\mathcal{W}$, and where α, β and ω, ω' are used as in the syntax of processes.

▶ **Definition 3.1** (Domain-aware Session Types). *The syntax of types* (A, B, C) *is defined by*

$$\begin{array}{lllllllll} A & ::= & \mathbf{1} & \mid & A \multimap B & \mid & A \otimes B & \mid & \&\{l_i : A_i\}_{i\in I} \;\mid\; \oplus\{l_i : A_i\}_{i\in I} \;\mid\; !A \\ & \mid & @_\omega\, A & \mid & \forall\alpha.A & \mid & \exists\alpha.A & \mid & \downarrow\alpha.A \end{array}$$

Types are the propositions of intuitionistic linear logic where the additives $A\&B$ and $A\oplus B$ are generalized to a labelled n-ary variant. Propositions take the standard interpretation as session types, extended with hybrid logic operators [5], with worlds interpreted as domains that are explicitly subject to an *accessibility relation* (in the style of [43]) that is tracked by environment Ω. Intuitively, Ω is made up of direct accessibility hypotheses of the form $\omega_1 \prec \omega_2$, meaning that domain ω_2 is accessible from ω_1.

Types are assigned to channel names; a *type assignment* $x{:}A[\omega]$ enforces the use of name x according to session A, *in the domain* ω. A *type environment* is a collection of type assignments. Besides the accessibility environment Ω just mentioned, our typing judgments consider two kinds of type environments: a *linear* part Δ and an *unrestricted* part Γ. They are subject to different structural properties: weakening and contraction principles hold for Γ but not for Δ. Empty environments are written as '$\cdot$'. We then consider two judgments:

$$\text{(i)}\quad \Omega \vdash \omega_1 \prec \omega_2 \qquad\text{and}\qquad \text{(ii)}\quad \Omega; \Gamma; \Delta \vdash P :: z{:}A[\omega]$$

Judgment (i) states that ω_1 can directly access ω_2 under the hypotheses in Ω. We write $\prec^*$ for the reflexive, transitive closure of $\prec$, and $\omega_1 \not\prec^* \omega_2$ when $\omega_1 \prec^* \omega_2$ does not hold. Judgment (ii) states that process P offers the session behavior specified by type A on channel z; the session s resides at domain ω, under the accessibility hypotheses Ω, using unrestricted sessions in Γ and linear sessions in Δ. Note that each hypothesis in Γ and Δ is labeled with a specific domain. We omit Ω when it is clear from context.

Typing Rules. Selected typing rules are given in Fig. 1; see [9] for the full listing. Right rules (marked with R) specify how to *offer* a session of a given type, left rules (marked with L) define how to *use* a session. The hybrid nature of the system induces a notion of *well-formedness* of sequents: a sequent $\Omega; \Gamma; \Delta \vdash P :: z : C[\omega_1]$ is *well-formed* if $\Omega \vdash \omega_1 \prec^* \omega_2$ for every $x{:}A[\omega_2] \in \Delta$, which we abbreviate as $\Omega \vdash \omega_1 \prec^* \Delta$, meaning that all domains mentioned in Δ are accessible from ω_1 (not necessarily in a single *direct* step). No such domain requirement is imposed on Γ. If an end sequent is well-formed, every sequent in its proof will also be well-formed. All rules (read bottom-up) preserve this invariant; only (cut), (copy), $(@\mathsf{R})$, $(\forall\mathsf{L})$ and $(\exists\mathsf{R})$ require explicit checks, which we discuss below. This invariant statically excludes interaction between sessions in accessible domains (cf. Theorem 3.7).

We briefly discuss some of the typing rules, first noting that we consider processes modulo structural congruence; hence, typability is closed under $\equiv$ by definition. Type $A \multimap B$ denotes a session that inputs a session of type A and proceeds as B. To offer $z{:}A \multimap B$ at domain ω, we input y along z that will offer A at ω and proceed, now offering $z{:}B$ at ω:

$$(\multimap\mathsf{R})\ \frac{\Omega; \Gamma; \Delta, y{:}A[\omega] \vdash P :: z{:}B[\omega]}{\Omega; \Gamma; \Delta \vdash z(y).P :: z{:}A \multimap B[\omega]} \qquad (\otimes\mathsf{R})\ \frac{\Omega; \Gamma; \Delta_1 \vdash P :: y{:}A[\omega] \quad \Omega; \Gamma; \Delta_2 \vdash Q :: z{:}B[\omega]}{\Omega; \Gamma; \Delta_1, \Delta_2 \vdash \overline{z}\langle y\rangle.(P \mid Q) :: z{:}A \otimes B[\omega]}$$

Dually, $A \otimes B$ denotes a session that outputs a session that will offer A and continue as B. To offer $z{:}A \otimes B$, we output a fresh name y with type A along z and proceed offering $z{:}B$.

The (cut) rule allows us to compose process P, which offers $x{:}A[\omega_2]$, with process Q, which uses $x{:}A[\omega_2]$ to offer $z{:}C[\omega_1]$. We require that domain ω_2 is accessible from ω_1 (i.e., $\omega_1 \prec^* \omega_2$). We also require $\omega_1 \prec^* \Delta_1$: the domains mentioned in Δ_1 (the context for P) must be accessible from ω_1, which follows from the transitive closure of the accessibility relation ($\prec^*$) using the intermediary domain ω_2. As in [10, 11], composition binds the name x:

$$(\mathsf{cut})\ \frac{\Omega \vdash \omega_1 \prec^* \omega_2 \quad \Omega \vdash \omega_1 \prec^* \Delta_1 \quad \Omega;\Gamma;\Delta_1 \vdash P :: x{:}A[\omega_2] \quad \Omega;\Gamma;\Delta_2, x{:}A[\omega_2] \vdash Q :: z{:}C[\omega_1]}{\Omega;\Gamma;\Delta_1,\Delta_2 \vdash (\boldsymbol{\nu} x)(P \mid Q) :: z{:}C[\omega_1]}$$

Type $\mathbf{1}$ means that no further interaction will take place on the session; names of type $\mathbf{1}$ may be passed around as opaque values. $\&\{l_i : A_i\}_{i\in I}$ types a session channel that offers its partner a choice between the A_i behaviors, each uniquely identified by a label l_i. Dually, $\oplus\{l_i : A_i\}_{i\in I}$ types a session that selects some behavior A_i by emitting the corresponding label. For flexibility and consistency with merge-based projectability in multiparty session types, rules for choice and selection induce a standard notion of session subtyping [26].

Type $!A$ types a shared (non-linear) channel, to be used by a server for spawning an arbitrary number of new sessions (possibly none), each one conforming to type A.

Following our previous remark on well-formed sequents, the only rules that appeal to accessibility are (@R), (@L), (copy), and (cut). These conditions are directly associated with varying degrees of flexibility in terms of typability, depending on what relationship is imposed between the domain to the left and to the right of the turnstile in the left rules. Notably, our system leverages the accessibility judgment to enforce that communication is only allowed between processes whose sessions are in (transitively) *accessible* domains.

The type operator $@_\omega$ realizes a *domain migration* mechanism which is specified both at the level of types and processes via name mobility tagged with a domain name. Thus, a channel typed with $@_{\omega_2}A$ denotes that behavior A is available by first *moving to* domain ω_2, directly accessible from the current domain. More precisely, we have:

$$(@\mathsf{R})\ \frac{\begin{array}{c}\Omega \vdash \omega_1 \prec \omega_2 \\ \Omega \vdash \omega_2 \prec^* \Delta \quad \Omega;\Gamma;\Delta \vdash P :: y{:}A[\omega_2]\end{array}}{\Omega;\Gamma;\Delta \vdash \overline{z}\langle y@\omega_2\rangle.P :: z{:}@_{\omega_2}A[\omega_1]} \qquad (@\mathsf{L})\ \frac{\Omega, \omega_2 \prec \omega_3;\Gamma;\Delta,\ y{:}A[\omega_3] \vdash P :: z{:}C[\omega_1]}{\Omega;\Gamma;\Delta, x{:}@_{\omega_3}A[\omega_2] \vdash x(y@\omega_3).P :: z{:}C[\omega_1]}$$

Hence, a process *offering* a behavior $z{:}@_{\omega_2}A$ at ω_1 ensures: (i) behavior A is available at ω_2 along a *fresh* session channel y that is emitted along z and (ii) ω_2 is directly accessible from ω_1. To maintain well-formedness of the sequent we also must check that all domains in Δ are still accessible from ω_2. Dually, *using* a service $x{:}@_{\omega_3}A[\omega_2]$ entails receiving a channel y that will offer behavior A at domain ω_3 (and also allowing the usage of the fact that $\omega_2 \prec \omega_3$).

Domain-quantified sessions introduce domains as *fresh* parameters to types: a particular service can be specified with the ability to refer to any existing directly accessible domain (via universal quantification) or to some *a priori* unspecified accessible domain:

$$(\forall\mathsf{R})\ \frac{\Omega, \omega_1 \prec \alpha;\Gamma;\Delta \vdash P :: z{:}A[\omega_1] \quad \alpha \notin \Omega,\Gamma,\Delta,\omega_1}{\Omega;\Gamma;\Delta \vdash z(\alpha).P :: z{:}\forall\alpha.A[\omega_1]} \qquad (\forall\mathsf{L})\ \frac{\begin{array}{c}\Omega \vdash \omega_2 \prec \omega_3 \\ \Omega;\Gamma;\Delta,\ x{:}A\{\omega_3/\alpha\}[\omega_2] \vdash Q :: z{:}C[\omega_1]\end{array}}{\Omega;\Gamma;\Delta, x{:}\forall\alpha.A[\omega_2] \vdash x\langle\omega_3\rangle.Q :: z{:}C[\omega_1]}$$

Rule (∀R) states that a process seeking to offer $\forall\alpha.A[\omega_1]$ denotes a service that is located at domain ω_1 but that may refer to any fresh domain directly accessible from ω_1 in its specification (e.g. through the use of @). Operationally, this means that the process must be ready to receive from its client a reference to the domain being referred to in the type, which is bound to α (occurring fresh in the typing derivation). Dually, Rule (∀L) indicates that a process interacting with a service of type $x{:}\forall\alpha.A[\omega_2]$ must make concrete the domain that is directly accessible from ω_2 it wishes to use, which is achieved by the appropriate output action. Rules (∃L) and (∃R) for the existential quantifier have a dual reading.

Finally, the type-level operator $\downarrow\alpha.A$ allows for a type to refer to its *current* domain:

$$(\downarrow\mathsf{R})\ \frac{\Omega; \Gamma; \Delta \vdash P :: z{:}A\{\omega/\alpha\}[\omega]}{\Omega; \Gamma; \Delta \vdash P :: z{:}\downarrow\alpha.A[\omega]} \qquad (\downarrow\mathsf{L})\ \frac{\Omega; \Gamma; \Delta, x{:}A\{\omega/\alpha\}[\omega] \vdash P :: z{:}C}{\Omega; \Gamma; \Delta, x{:}\downarrow\alpha.A[\omega] \vdash P :: z{:}C}$$

The typing rules that govern $\downarrow\alpha.A$ are completely symmetric and produce no action at the process level, merely instantiating the domain variable α with the current domain ω of the session. As will be made clear in § 4, this connective plays a crucial role in ensuring the correctness of our analysis of multiparty domain-aware sessions in our logical setting.

By developing our type theory with an explicit domain accessibility judgment, we can consider the accessibility relation as a *parameter* of the framework. This allows changing accessibility relations and their properties without having to alter the entire system. To consider the simplest possible accessibility relation, the only defining rule for accessibility would be Rule (whyp) in Fig. 1. To consider an accessibility relation which is an equivalence relation we would add reflexivity, transitivity, and symmetry rules to the judgment.

Discussion and Examples. Being an interpretation of *hybridized* linear logic, our domain-aware theory is *conservative* wrt the Curry-Howard interpretation of session types in [10, 11], in the following sense: the system in [10, 11] corresponds to the case where every session resides at the same domain. As in [10, 11], the sequent calculus for the underlying (hybrid) linear logic can be recovered from our typing rules by erasing processes and name assignments.

Conversely, a fundamental consequence of our hybrid interpretation is that it *refines* the session type structure in non-trivial ways. By requiring that communication only occurs between sessions located at the same (or accessible) domain we effectively introduce a new layer of reasoning to session type systems. To illustrate this feature, consider the following session type WStore, which specifies a simple interaction between a web store and its clients:

$$\mathsf{WStore} \triangleq \mathsf{addCart} \multimap \&\{\mathit{buy} : \mathsf{Pay}\,,\ \mathit{quit} : \mathbf{1}\} \qquad \mathsf{Pay} \triangleq \mathsf{CCNum} \multimap \oplus\{\mathit{ok} : \mathsf{Rcpt} \otimes \mathbf{1}\,,\ \mathit{nok} : \mathbf{1}\}$$

WStore allows clients to checkout their shopping carts by emitting a *buy* message or to *quit*. In the former case, the client pays for the purchase by sending their credit card data. If a banking service (not shown) approves the transaction (via an *ok* message), a receipt is emitted. Representable in existing session type systems (e.g. [10, 47, 31]), types WStore and Pay describe the intended communications but fail to capture the crucial fact that in practice the client's sensitive information should only be requested after entering a secure domain. To address this limitation, we can use type-level domain migration to *refine* WStore and Pay:

$$\begin{aligned} \mathsf{WStore}_{\mathsf{sec}} &\triangleq \quad \mathsf{addCart} \multimap \&\{\mathit{buy} : @_{\mathsf{sec}}\,\mathsf{Pay}_{\mathsf{bnk}}, \mathtt{quit} : \mathbf{1}\} \\ \mathsf{Pay}_{\mathsf{bnk}} &\triangleq \quad \mathsf{CCNum} \multimap \oplus\{\mathit{ok} : (@_{\mathsf{bnk}}\mathsf{Rcpt}) \otimes \mathbf{1}, \mathit{nok} : \mathbf{1}\} \end{aligned}$$

$\mathsf{WStore}_{\mathsf{sec}}$ decrees that the interactions pertinent to type $\mathsf{Pay}_{\mathsf{bnk}}$ should be preceded by a migration step to the trusted domain $\mathtt{sec}$, which should be directly accessible from $\mathsf{WStore}_{\mathsf{sec}}$'s current domain. The type also specifies that the receipt must originate from a bank domain $\mathtt{bnk}$ (e.g., ensuring that the receipt is never produced by the store without entering $\mathtt{bnk}$). When considering the interactions with a client (at domain $\mathtt{c}$) that checks out their cart, we reach a state that is typed with the following judgment:

$$\mathtt{c} \prec \mathtt{ws}; \cdot; x{:}@_{\mathsf{sec}}\mathsf{Pay}_{\mathsf{bnk}}[\mathtt{ws}] \vdash \mathit{Client} :: z{:}@_{\mathsf{sec}}\mathbf{1}[\mathtt{c}]$$

At this point, it is *impossible* for a (typed) client to interact with the behavior that is protected by the domain $\mathtt{sec}$, since it is not the case that $\mathtt{c} \prec^* \mathtt{sec}$. That is, no judgment of the form $\mathtt{c} \prec \mathtt{ws}; \cdot; \mathsf{Pay}_{\mathsf{bnk}}[\mathtt{sec}] \vdash \mathit{Client}' :: z{:}T[\mathtt{c}]$ is derivable. This ensures, e.g., that a

$$(\mathsf{whyp})\ \frac{}{\Omega, \omega_1 \prec \omega_2 \vdash \omega_1 \prec \omega_2} \qquad (\mathsf{id})\ \frac{}{\Omega; \Gamma; x{:}A[\omega] \vdash [x \leftrightarrow z] :: z{:}A[\omega]}$$

$$(\mathsf{@R})\ \frac{\Omega \vdash \omega_1 \prec \omega_2 \quad \Omega \vdash \omega_2 \prec^* \Delta \quad \Omega; \Gamma; \Delta \vdash P :: y{:}A[\omega_2]}{\Omega; \Gamma; \Delta \vdash \overline{z}\langle y@\omega_2\rangle.P :: z{:}@_{\omega_2}A[\omega_1]} \qquad (\mathsf{@L})\ \frac{\Omega, \omega_2 \prec \omega_3; \Gamma; \Delta,\ y{:}A[\omega_3] \vdash P :: z{:}C[\omega_1]}{\Omega; \Gamma; \Delta, x{:}@_{\omega_3}A[\omega_2] \vdash x(y@\omega_3).P :: z{:}C[\omega_1]}$$

$$(\forall\mathsf{R})\ \frac{\Omega, \omega_1 \prec \alpha; \Gamma; \Delta \vdash P :: z{:}A[\omega_1] \quad \alpha \notin \Omega, \Gamma, \Delta, \omega_1}{\Omega; \Gamma; \Delta \vdash z(\alpha).P :: z{:}\forall\alpha.A[\omega_1]} \qquad (\forall\mathsf{L})\ \frac{\Omega \vdash \omega_2 \prec \omega_3 \quad \Omega; \Gamma; \Delta,\ x{:}A\{\omega_3/\alpha\}[\omega_2] \vdash Q :: z{:}C[\omega_1]}{\Omega; \Gamma; \Delta, x{:}\forall\alpha.A[\omega_2] \vdash x\langle\omega_3\rangle.Q :: z{:}C[\omega_1]}$$

$$(\exists\mathsf{R})\ \frac{\Omega \vdash \omega_1 \prec \omega_2 \quad \Omega; \Gamma; \Delta \vdash P :: z{:}A\{\omega_2/\alpha\}[\omega_1]}{\Omega; \Gamma; \Delta \vdash z\langle\omega_2\rangle.P :: z{:}\exists\alpha.A[\omega_1]} \qquad (\exists\mathsf{L})\ \frac{\Omega, \omega_2 \prec \alpha; \Gamma; \Delta,\ x{:}A[\omega_2] \vdash Q :: z{:}C[\omega_1]}{\Omega; \Gamma; \Delta, x{:}\exists\alpha.A[\omega_2] \vdash x(\alpha).Q :: z{:}C[\omega_1]}$$

$$(\downarrow\mathsf{R})\ \frac{\Omega; \Gamma; \Delta \vdash P :: z{:}A\{\omega/\alpha\}[\omega]}{\Omega; \Gamma; \Delta \vdash P :: z{:}{\downarrow}\alpha.A[\omega]} \qquad (\downarrow\mathsf{L})\ \frac{\Omega; \Gamma; \Delta, x{:}A\{\omega/\alpha\}[\omega] \vdash P :: z{:}C}{\Omega; \Gamma; \Delta, x{:}{\downarrow}\alpha.A[\omega] \vdash P :: z{:}C}$$

$$(\mathsf{copy})\ \frac{\Omega \vdash \omega_1 \prec^* \omega_2 \quad \Omega; \Gamma, u{:}A[\omega_2]; \Delta, y{:}A[\omega_2] \vdash P :: z{:}C[\omega_1]}{\Omega; \Gamma, u{:}A[\omega_2]; \Delta \vdash \overline{u}\langle y\rangle.P :: z{:}C[\omega_1]}$$

$$(\mathsf{cut})\ \frac{\Omega \vdash \omega_1 \prec^* \omega_2 \quad \Omega \vdash \omega_2 \prec^* \Delta_1 \quad \Omega; \Gamma; \Delta_1 \vdash P :: x{:}A[\omega_2] \quad \Omega; \Gamma; \Delta_2, x{:}A[\omega_2] \vdash Q :: z{:}C[\omega_1]}{\Omega; \Gamma; \Delta_1, \Delta_2 \vdash (\boldsymbol{\nu}x)(P \mid Q) :: z{:}C[\omega_1]}$$

Figure 1 Typing Rules (Excerpt – see [9]).

client cannot exploit the payment platform of the web store by accessing the trusted domain in unforeseen ways. The client can only communicate in the secure domain *after* the web store service has migrated accordingly, as shown by the judgment

$$\mathtt{c} \prec \mathtt{ws}, \mathtt{ws} \prec \mathtt{sec}; \cdot; x'{:}\mathsf{Pay}_{\mathtt{bnk}}[\mathtt{sec}] \vdash \mathit{Client}' :: z'{:}\mathbf{1}[\mathtt{sec}].$$

Technical Results. We state the main results of type safety via type preservation (Theorem 3.3) and global progress (Theorem 3.4). These results directly ensure session fidelity and deadlock-freedom. Typing also ensures termination, i.e., processes do not exhibit infinite reduction paths (Theorem 3.5). We note that in the presence of termination, our progress result ensures that communication actions are always guaranteed to take place. Moreover, as a property specific to domain-aware processes, we show *domain preservation*, i.e., processes respect their domain accessibility conditions (Theorem 3.7). The formal development of these results relies on a *domain-aware* labeled transition system [9], defined as a simple generalization of the early labelled transition system for the session π-calculus given in [10, 11].

Type Safety and Termination. Following [10, 11], our proof of type preservation relies on a simulation between reductions in the session-typed π-calculus and logical proof reductions.

▶ **Lemma 3.2** (Domain Substitution). *Suppose* $\Omega \vdash \omega_1 \prec \omega_2$. *Then we have:*

- *If* $\Omega, \omega_1 \prec \alpha, \Omega'; \Gamma; \Delta \vdash P :: z{:}A[\omega]$ *then* $\Omega, \Omega'\{\omega_2/\alpha\}; \Gamma\{\omega_2/\alpha\}; \Delta\{\omega_2/\alpha\} \vdash P\{\omega_2/\alpha\} :: z{:}A[\omega\{\omega_2/\alpha\}]$.
- $\Omega, \alpha \prec \omega_2, \Omega'; \Gamma; \Delta \vdash P :: z{:}A[\omega]$ *then* $\Omega, \Omega'\{\omega_1/\alpha\}; \Gamma\{\omega_1/\alpha\}; \Delta\{\omega_1/\alpha\} \vdash P\{\omega_1/\alpha\} :: z{:}A[\omega\{\omega_1/\alpha\}]$.

Safe domain communication relies on domain substitution preserving typing (Lemma 3.2).

▶ **Theorem 3.3** (Type Preservation). *If* $\Omega; \Gamma; \Delta \vdash P :: z{:}A[\omega]$ *and* $P \to Q$ *then* $\Omega; \Gamma; \Delta \vdash Q :: z{:}A[\omega]$.

Proof (Sketch). The proof mirrors those of [10, 11, 8, 44], relying on a series of lemmas relating the result of dual process actions (via our LTS semantics) with typable parallel compositions through the (cut) rule [9]. For session type constructors of [10], the results are unchanged. For the domain-aware session type constructors, the development is identical that of [8] and [44], which deal with communication of types and data terms, respectively. ◀

Following [10, 11], the proof of global progress relies on a notion of a *live* process, which intuitively consists of a process that has not yet fully carried out its ascribed session behavior, and thus is a parallel composition of processes where at least one is a non-replicated process, guarded by some action. Formally, we define $live(P)$ if and only if $P \equiv (\boldsymbol{\nu}\tilde{n})(\pi.Q \mid R)$, for some R, names $\tilde{n}$ and a non-replicated guarded process $\pi.Q$.

▶ **Theorem 3.4** (Global Progress). *If $\Omega; \cdot; \cdot \vdash P :: x{:}\mathbf{1}[\omega]$ and $live(P)$ then $\exists Q$ s.t. $P \rightarrow Q$.*

Note that Theorem 3.4 is without loss of generality since using the cut rules we can compose arbitrary well-typed processes together and x need not occur in P due to Rule ($\mathbf{1}$R).

Termination (strong normalization) is a relevant property for interactive systems: while from a global perspective they are meant to run forever, at a local level participants should always react within a finite amount of time, and never engage into infinite internal behavior. We say that a process P *terminates*, noted $P \Downarrow$, if there is no infinite reduction path from P.

▶ **Theorem 3.5** (Termination). *If $\Omega; \Gamma; \Delta \vdash P :: x{:}A[\omega]$ then $P \Downarrow$.*

Proof (Sketch). By adapting the *linear* logical relations given in [38, 39, 8]. For the system in § 3 without quantifiers, the logical relations correspond to those in [38, 39], extended to carry over Ω. When considering quantifiers, the logical relations resemble those proposed for polymorphic session types in [8], noting that no impredicativity concerns are involved. ◀

Domain Preservation. As a consequence of the hybrid nature of our system, well-typed processes are guaranteed not only to faithfully perform their prescribed behavior in a deadlock-free manner, but they also do so without breaking the constraints put in place on domain accessibility given by our well-formedness constraint on sequents.

▶ **Theorem 3.6.** *Let $\mathcal{E}$ be a derivation of $\Omega; \Gamma; \Delta \vdash P :: z{:}A[\omega]$. If $\Omega; \Gamma; \Delta \vdash P :: z{:}A[\omega]$ is well-formed then every sub-derivation in $\mathcal{E}$ well-formed.*

While inaccessible domains can appear in Γ, such channels can never be used and thus can not appear in a well-typed process due to the restriction on the (copy) rule. Combining Theorems 3.3 and 3.6 we can then show that even if a session in the environment changes domains, typing ensures that such a domain will be (transitively) accessible:

▶ **Theorem 3.7.** *Let (1) $\Omega; \Gamma; \Delta, \Delta' \vdash (\boldsymbol{\nu}x)(P \mid Q) :: z : A[\omega]$, (2) $\Omega; \Gamma; \Delta \vdash P :: x{:}B[\omega'']$, and (3) $\Omega; \Gamma; \Delta', x{:}B[\omega'] \vdash Q :: z{:}A[\omega]$. If $(\boldsymbol{\nu}x)(P \mid Q) \rightarrow (\boldsymbol{\nu}x)(P' \mid Q')$ then: (a) $\Omega; \Gamma; \Delta \vdash P' :: x'{:}B'[\omega'']$, for some x', B', ω''; (b) $\Omega; \Gamma, \Delta', x'{:}B'[\omega''] \vdash Q' :: z{:}A[\omega]$; (c) $\omega \prec^* \omega''$.*

4 Domain-Aware Multiparty Session Types

We now shift our attention to multiparty session types [32]. We consider the standard ingredients: *global types*, *local types*, and the *projection function* that connects the two. Our global types include a new domain-aware construct, p moves $\widetilde{q}$ to ω for $G_1 \,;\, G_2$; our local types exploit the hybrid session types from Def. 3.1. Rather than defining a separate type system based on local types for the process model of § 2, our analysis of multiparty protocols extends the approach defined in [7], which uses *medium processes* to characterize correct multiparty implementations. The advantages are twofold: on the one hand, medium processes provide a precise semantics for global types; on the other hand, they enable the principled transfer of the correctness properties established in § 3 for binary sessions (type preservation, global progress, termination, domain preservation) to the multiparty setting. Below, *participants* are ranged over by $\mathtt{p}, \mathtt{q}, \mathtt{r}, \ldots$; we write $\widetilde{\mathtt{q}}$ to denote a finite set of participants $\mathtt{q}_1, \ldots, \mathtt{q}_n$.

Besides the new domain-aware global type, our syntax of global types includes constructs from [32, 21]. We consider value passing in branching (cf. U below), fully supporting delegation. To streamline the presentation, we consider global types without recursion.

▶ **Definition 4.1** (Global and Local Types). *Define global types (G) and local types (T) as*

$$\begin{array}{rl} U ::= & \mathsf{bool} \mid \mathsf{nat} \mid \mathsf{str} \mid \ldots \mid T \\ G ::= & \mathsf{end} \mid \mathsf{p} \twoheadrightarrow \mathsf{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I} \mid \mathsf{p}\,\mathsf{moves}\,\widetilde{q}\,\mathsf{to}\,\omega\,\mathsf{for}\,G_1\,;\,G_2 \\ T ::= & \mathsf{end} \mid \mathsf{p}?\{l_i\langle U_i\rangle.T_i\}_{i\in I} \mid \mathsf{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I} \mid \forall\alpha.T \mid \exists\alpha.T \mid @_\alpha T \mid \downarrow\alpha.T \end{array}$$

The completed global type is denoted $\mathtt{end}$. Given a finite I and pairwise different labels, $\mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I}$ specifies that by choosing label l_i, participant $\mathtt{p}$ may send a message of type U_i to participant $\mathtt{q}$, and then continue as G_i. We decree $\mathtt{p} \neq \mathtt{q}$, so reflexive interactions are disallowed. The global type $\mathsf{p}\,\mathsf{moves}\,\widetilde{\mathsf{q}}\,\mathsf{to}\,\omega\,\mathsf{for}\,G_1\,;\,G_2$ specifies the migration of participants $\mathtt{p}, \widetilde{\mathtt{q}}$ to domain ω in order to perform the *sub-protocol* G_1; this migration is lead by $\mathtt{p}$. Subsequently, all of $\mathtt{p}, \widetilde{\mathtt{q}}$ migrate from ω back to their original domains and protocol G_2 is executed. This intuition will be made precise by the medium processes for global types (cf. Def. 4.8). Notice that G_1 and G_2 may involve different sets of participants. In writing $\mathsf{p}\,\mathsf{moves}\,\widetilde{\mathsf{q}}\,\mathsf{to}\,\omega\,\mathsf{for}\,G_1\,;\,G_2$ we assume two natural conditions: (a) all migrating participants intervene in the sub-protocol (i.e., the set of participants of G_1 is exactly $\mathtt{p}, \widetilde{\mathtt{q}}$) and (b) domain ω is accessible (via $\prec$) by all these migrating participants in G_1. While subprotocols and session delegation may appear as similar, delegation supports a different idiom altogether, and has no support for domain awareness. Unlike delegation, with subprotocols we can specify a point where some of the participants perform a certain protocol *within the same multiparty session* and then return to the main session as an ensemble.

▶ **Definition 4.2.** *The set of participants of G (denoted* $\mathsf{part}(G)$*) is defined as:* $\mathsf{part}(\mathsf{end}) = \emptyset$, $\mathsf{part}(\mathsf{p} \twoheadrightarrow \mathsf{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I}) = \{\mathsf{p},\mathsf{q}\} \cup \bigcup_{i\in I} \mathsf{part}(G_i)$, $\mathsf{part}(\mathsf{p}\,\mathsf{moves}\,\widetilde{\mathsf{q}}\,\mathsf{to}\,\omega\,\mathsf{for}\,G_1\,;\,G_2) = \{\mathsf{p}\} \cup \widetilde{\mathsf{q}} \cup \mathsf{part}(G_1) \cup \mathsf{part}(G_2)$. *We sometimes write* $\mathsf{p} \in G$ *to mean* $\mathsf{p} \in \mathsf{part}(G)$.

Global types are projected onto participants so as to obtain local types. The terminated local type is $\mathtt{end}$. The local type $\mathtt{p}?\{l_i\langle U_i\rangle.T_i\}_{i\in I}$ denotes an offer of a set of labeled alternatives; the local type $\mathtt{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I}$ denotes a behavior that chooses one of such alternatives. Exploiting the domain-aware framework in § 3, we introduce four new local types. They increase the expressiveness of standard local types by specifying universal and existential quantification over domains ($\forall\alpha.T$ and $\exists\alpha.T$), migration to a specific domain ($@_\alpha T$), and a reference to the current domain ($\downarrow\alpha.T$, with α occurring in T).

We now define *(merge-based) projection* for global types [21]. To this end, we rely on a *merge* operator on local types, which in our case considers messages U.

▶ **Definition 4.3** (Merge). *We define* $\sqcup$ *as the commutative partial operator on base and local types such that* $\mathsf{bool} \sqcup \mathsf{bool} = \mathsf{bool}$ *(and analogously for other base types), and*

1. $T \sqcup T = T$*, where* T *is one of the following:* end, $\mathsf{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I}$, $@_\omega T$, $\forall\alpha.T$, *or* $\exists\alpha.T$*;*
2. $\mathsf{p}?\{l_k\langle U_k\rangle.T_k\}_{k\in K} \sqcup \mathsf{p}?\{l'_j\langle U'_j\rangle.T'_j\}_{j\in J} =$
   $\mathsf{p}?\big(\{l_k\langle U_k\rangle.T_k\}_{k\in K\setminus J} \cup \{l'_j\langle U'_j\rangle.T'_j\}_{j\in J\setminus K} \cup \{l_l\langle U_l \sqcup U'_l\rangle.(T_l \sqcup T'_l)\}_{l\in K\cap J}\big)$

and is undefined otherwise.

Therefore, for $U_1 \sqcup U_2$ to be defined there are two options: (a) U_1 and U_2 are identical base, terminated, selection, or "hybrid" local types; (b) U_1 and U_2 are branching types, but not necessarily identical: they may offer different options but with the condition that the behavior in labels occurring in both U_1 and U_2 must be mergeable.

To define projection and medium processes for the global type $\mathsf{p}\,\mathsf{moves}\,\widetilde{\mathsf{q}}\,\mathsf{to}\,\omega\,\mathsf{for}\,G_1\,;\,G_2$, we require ways of "fusing" local types and processes. The intent is to capture in a single (sequential) specification the behavior of two distinct (sequential) specifications, i.e., those corresponding to protocols G_1 and G_2. For local types, we have the following definition, which safely appends a local type to another:

▶ **Definition 4.4** (Local Type Fusion). *The* fusion *of T_1 and T_2, written $T_1 \circ T_2$, is given by:*

$$\begin{array}{lcl@{\qquad}lcl}
\mathsf{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I} \circ T & = & \mathsf{p}!\{l_i\langle U_i\rangle.(T_i \circ T)\}_{i\in I} & \mathsf{end} \circ T & = & T \\
\mathsf{p}?\{l_i\langle U_i\rangle.T_i\}_{i\in I} \circ T & = & \mathsf{p}?\{l_i\langle U_i\rangle.(T_i \circ T)\}_{i\in I} & (\exists\alpha.T_1) \circ T & = & \exists\alpha.(T_1 \circ T) \\
(\forall\alpha.T_1) \circ T & = & \forall\alpha.(T_1 \circ T) & (@_\alpha T_1) \circ T & = & @_\alpha (T_1 \circ T) \\
(\downarrow\alpha.T_1) \circ T & = & \downarrow\alpha.(T_1 \circ T) & & &
\end{array}$$

This way, e.g., if $T_1 = \exists\alpha.@_\alpha\,\mathsf{p}?\{l_1\langle\mathsf{Int}\rangle.\mathsf{end}\,,\, l_2\langle\mathsf{Bool}\rangle.\mathsf{end}\}$ and $T_2 = @_\omega\,\mathsf{q}!\{l\langle\mathsf{Str}\rangle.\mathsf{end}\}$, then $T_1 \circ T_2 = \exists\alpha.@_\alpha\,\mathsf{p}?\{l_1\langle\mathsf{Int}\rangle.@_\omega\,\mathsf{q}!\{l\langle\mathsf{Str}\rangle.\mathsf{end}\}\,,\, l_2\langle\mathsf{Bool}\rangle.@_\omega\,\mathsf{q}!\{l\langle\mathsf{Str}\rangle.\mathsf{end}\}\}$. We can now define:

▶ **Definition 4.5** (Merge-based Projection [21]). *Let G be a global type. The* merge-based projection *of G under participant $\mathtt{r}$, denoted $G{\upharpoonright}\mathtt{r}$, is defined as $\mathsf{end}{\upharpoonright}\mathtt{r} = \mathsf{end}$ and*

- $$\mathsf{p} \twoheadrightarrow \mathsf{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i\in I}{\upharpoonright}\mathtt{r} = \begin{cases} \mathsf{p}!\{l_i\langle U_i\rangle.G_i{\upharpoonright}\mathtt{r}\}_{i\in I} & \textit{if } \mathtt{r} = \mathtt{p} \\ \mathsf{p}?\{l_i\langle U_i\rangle.G_i{\upharpoonright}\mathtt{r}\}_{i\in I} & \textit{if } \mathtt{r} = \mathtt{q} \\ \sqcup_{i\in I}\, G_i{\upharpoonright}\mathtt{r} & \textit{otherwise } (\sqcup \textit{ as in Def. 4.3}) \end{cases}$$
- $$(\mathsf{p}\,\mathsf{moves}\,\widetilde{\mathsf{q}}\,\mathsf{to}\,\omega\,\mathsf{for}\,G_1\,;\,G_2){\upharpoonright}\mathtt{r} = \begin{cases} \downarrow\beta.(\exists\alpha.@_\alpha\,G_1{\upharpoonright}\mathtt{r}) \circ @_\beta\,G_2{\upharpoonright}\mathtt{r} & \textit{if } \mathtt{r} = \mathtt{p} \\ \downarrow\beta.(\forall\alpha.@_\alpha\,G_1{\upharpoonright}\mathtt{r}) \circ @_\beta\,G_2{\upharpoonright}\mathtt{r} & \textit{if } \mathtt{r} \in \widetilde{\mathtt{q}} \\ G_2{\upharpoonright}\mathtt{r} & \textit{otherwise} \end{cases}$$

When no side condition holds, the map is undefined.

The projection for the type $\mathsf{p}\,\mathsf{moves}\,\widetilde{\mathsf{q}}\,\mathsf{to}\,w\,\mathsf{for}\,G_1\,;\,G_2$ is one of the key points in our analysis. The local type for $\mathtt{p}$, the leader of the migration, starts by binding the identity of its current domain (say, $\omega_{\mathtt{p}}$) to β. Then, the (fresh) domain ω is communicated, and there is a migration step to ω, which is where protocol $G_1{\upharpoonright}\mathtt{p}$ will be performed. Finally, there is a migration step from ω back to $\omega_{\mathtt{p}}$; once there, the protocol $G_2{\upharpoonright}\mathtt{p}$ will be performed. The local type for all of $\mathsf{q}_i \in \widetilde{\mathsf{q}}$ follows accordingly: they expect ω from $\mathtt{p}$; the migration from their original domains to ω (and back) is as for $\mathtt{p}$. For participants in G_1, the fusion on local types (Def. 4.4) defines a local type that includes the actions for G_1 but also for G_2, if any: a participant in G_1 need not be involved in G_2. Interestingly, the resulting local types $\downarrow\beta.(\exists\alpha.@_\alpha\,G_1{\upharpoonright}\mathtt{p}) \circ @_\beta\,G_2{\upharpoonright}\mathtt{p}$ and $\downarrow\beta.(\forall\alpha.@_\alpha\,G_1{\upharpoonright}\mathsf{q}_i) \circ @_\beta\,G_2{\upharpoonright}\mathsf{q}_i$ define a precise combination of hybrid connectives whereby each migration step is bound by a quantifier or the current domain.

The following notion of *well-formedness* for global types is standard:

▶ **Definition 4.6** (Well-Formed Global Types [32]). *We say that global type G is* well-formed (WF, in the following) *if the projection $G{\upharpoonright}\mathtt{r}$ is defined for all $\mathtt{r} \in G$.*

Analyzing Global Types via Medium Processes. A *medium process* is a well-typed process from § 2 that captures the communication behavior of the domain-aware global types of Def. 4.1. Here we define medium processes and establish two fundamental characterization results for them (Theorems 4.11 and 4.12). We shall consider names *indexed by participants*:

given a name c and a participant $\mathtt{p}$, we use $c_{\mathtt{p}}$ to denote the name along which the session behavior of $\mathtt{p}$ will be made available. This way, if $\mathtt{p} \neq \mathtt{q}$ then $c_{\mathtt{p}} \neq c_{\mathtt{q}}$. To define mediums, we need to append or fuse sequential processes, just as Def. 4.4 fuses local types:

▶ **Definition 4.7** (Fusion of Processes). *We define $\circ$ as the partial operator on well-typed processes such that (with $\pi \in \{c(y), c\langle\omega\rangle, c(\alpha), c\langle y@\omega\rangle, c(y@\omega), c \triangleleft l\}$) :*

$$\begin{array}{rcl@{\qquad}rcl} c\langle y\rangle.([u \leftrightarrow y] \mid P) \circ Q & \triangleq & c\langle y\rangle.([u \leftrightarrow y] \mid (P \circ Q)) & \mathbf{0} \circ Q & \triangleq & Q \\ c \triangleright \big\{l_i : P_i\big\}_{i \in I} \circ Q & \triangleq & c \triangleright \big\{l_i : (P_i \circ Q)\big\}_{i \in I} & (\pi.P) \circ Q & \triangleq & \pi.(P \circ Q) \end{array}$$

and is undefined otherwise.

The previous definition suffices to define a medium process (or simply *medium*), which uses indexed names to uniformly capture the behavior of a global type:

▶ **Definition 4.8** (Medium Process). *Let G be a global type (cf. Def. 4.1), $\tilde{c}$ be a set of indexed names, and $\tilde{\omega}$ a set of domains. The* medium *of G, denoted $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$, is defined as:*

$$\begin{cases} \mathbf{0} & \textit{if } G = \mathsf{end} \\ c_{\mathtt{p}} \triangleright \big\{l_i : c_{\mathtt{p}}(u).c_{\mathtt{q}} \triangleleft l_i; \overline{c_{\mathtt{q}}}\langle v\rangle.([u \leftrightarrow v] \mid \mathsf{M}^{\tilde{\omega}}[\![G_i]\!](\tilde{c}))\big\}_{i \in I} & \textit{if } G = \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i \in I} \\ c_{\mathtt{p}}(\alpha).c_{\mathtt{q}_1}\langle\alpha\rangle.\cdots.c_{\mathtt{q}_n}\langle\alpha\rangle. & \textit{if } G = \mathtt{p}\,\mathsf{moves}\,\mathtt{q}_1, \ldots, \mathtt{q}_n\,\mathsf{to}\,w\,\mathsf{for}\,G_1\,;\,G_2 \\ \quad c_{\mathtt{p}}(y_{\mathtt{p}}@\alpha).c_{\mathtt{q}_1}(y_{\mathtt{q}_1}@\alpha).\cdots.c_{\mathtt{q}_n}(y_{\mathtt{q}_n}@\alpha). & \\ \qquad \mathsf{M}^{\tilde{\omega}\{\alpha/\omega_{\mathtt{p}}, \ldots, \alpha/\omega_{\mathtt{q}_n}\}}[\![G_1]\!](\tilde{y}) \circ & \\ \qquad\quad (y_{\mathtt{p}}(m_{\mathtt{p}}@\omega_{\mathtt{p}}).y_{\mathtt{q}_1}(m_{\mathtt{q}_1}@\omega_{\mathtt{q}_1}).\cdots.y_{\mathtt{q}_n}(m_{\mathtt{q}_n}@\omega_{\mathtt{q}_n}). & \\ \qquad\qquad \mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{m})) & \end{cases}$$

where $\mathsf{M}^{\tilde{\omega}}[\![G_1]\!](\tilde{c}) \circ \mathsf{M}^{\tilde{\omega}}[\![G_2]\!](\tilde{c})$ is as in Def. 4.7.

The medium for $G = \mathtt{p} \twoheadrightarrow \mathtt{q}{:}\{l_i\langle U_i\rangle.G_i\}_{i \in I}$ exploits four prefixes to mediate in the interaction between the implementations of $\mathtt{p}$ and $\mathtt{q}$: the first two prefixes (on name $c_{\mathtt{p}}$) capture the label selected by $\mathtt{p}$ and the subsequently received value; the third and fourth prefixes (on name $c_{\mathtt{q}}$) propagate the choice and forward the value sent by $\mathtt{p}$ to $\mathtt{q}$. We omit the forwarding and value exchange when the interaction does not involve a value payload.

The medium for $G = \mathtt{p}\,\mathsf{moves}\,\mathtt{q}_1, \ldots, \mathtt{q}_n\,\mathsf{to}\,w\,\mathsf{for}\,G_1\,;\,G_2$ showcases the expressivity and convenience of our domain-aware process framework. In this case, the medium's behavior takes place through the following steps: First, $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$ inputs a domain identifier (say, ω) from $\mathtt{p}$ which is forwarded to $\mathtt{q}_1, \ldots, \mathtt{q}_n$, the other participants of G_1. Secondly, the roles $\mathtt{p}, \mathtt{q}_1, \ldots, \mathtt{q}_n$ migrate from their domains $\omega_{\mathtt{p}}, \omega_{\mathtt{q}_1} \ldots, \omega_{\mathtt{q}_n}$ to ω. At this point, the medium for G_1 can execute, keeping track the current domain ω for all participants. Finally, the participants of G_1 migrate back to their original domains and the medium for G_2 executes.

Recalling the domain-aware global type of § 1, we produce its medium process:

$$\begin{array}{l} c_{\mathtt{cl}} \triangleright \big\{\mathit{request} : c_{\mathtt{cl}}(r).c_{\mathtt{mw}} \triangleleft \mathit{request}; \overline{c_{\mathtt{mw}}}\langle v\rangle.([r \leftrightarrow v] \mid \\ \quad c_{\mathtt{mw}} \triangleright \big\{\mathit{reply} : c_{\mathtt{mw}}(a).c_{\mathtt{cl}} \triangleleft \mathit{reply}; \overline{c_{\mathtt{cl}}}\langle n\rangle.([a \leftrightarrow n] \mid c_{\mathtt{mw}} \triangleright \big\{\mathit{done} : c_{\mathtt{serv}} \triangleleft \mathit{done}; \mathbf{0}\big\}), \\ \qquad\quad \mathit{wait} : c_{\mathtt{cl}} \triangleleft \mathit{wait}; c_{\mathtt{mw}} \triangleright \big\{\mathit{init} : c_{\mathtt{serv}} \triangleleft \mathit{init}; c_{\mathtt{mw}}(w_{\mathtt{priv}}).c_{\mathtt{serv}}\langle w_{\mathtt{priv}}\rangle. \\ \qquad\qquad\qquad\qquad\qquad c_{\mathtt{mw}}(y_{\mathtt{mw}}@w_{\mathtt{priv}}).c_{\mathtt{serv}}(y_{\mathtt{serv}}@w_{\mathtt{priv}}).\mathsf{M}^{\tilde{w_{\mathtt{priv}}}}[\![\mathit{Offload}]\!](y_{\mathtt{mw}}, y_{\mathtt{serv}}) \circ \\ \qquad\qquad\qquad\qquad\qquad (y_{\mathtt{mw}}(z_{\mathtt{mw}}@w_{\mathtt{mw}}).y_{\mathtt{serv}}(z_{\mathtt{serv}}@w_{\mathtt{serv}}). \\ \qquad\qquad\qquad\qquad\qquad z_{\mathtt{mw}} \triangleright \big\{\mathit{reply} : z_{\mathtt{mw}}(a).c_{\mathtt{cl}} \triangleleft \mathit{reply}; \overline{c_{\mathtt{cl}}}\langle n\rangle.([a \leftrightarrow n] \mid \mathbf{0})\big\})\big\}\big\})\big\} \end{array}$$

The medium ensures the client's domain remains fixed through the entire interaction, regardless of whether the middleware chooses to interact with the server. This showcases how our medium transparently manages domain migration of participants.

Characterization Results. We state results that offer a sound and complete account of the relationship between: (i) a global type G (and its local types), (ii) its medium process $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$, and (iii) process implementations for the participants $\{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$ of G. In a nutshell, these results say that the typeful composition of $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$ with processes for each $\mathtt{p}_1, \ldots, \mathtt{p}_n$ (well-typed in the system of § 3) performs the intended global type. Crucially, these processes reside in distinct domains and can be independently developed, guided by their local type – they need not know about the medium's existence or structure. The results generalize those in [7] to the domain-aware setting. Given a global type G with $\mathsf{part}(G) = \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$, below we write $\mathsf{npart}(G)$ to denote the set of indexed names $\{c_{\mathtt{p}_1}, \ldots, c_{\mathtt{p}_n}\}$. We define:

▶ **Definition 4.9** (Compositional Typing). *We say* $\Omega; \Gamma; \Delta \vdash \mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) :: z{:}C$ *is a* compositional typing *if: (i) it is a valid typing derivation; (ii)* $\mathsf{npart}(G) \subseteq \mathit{dom}(\Delta)$*; and (iii)* $C = \mathbf{1}$.

A compositional typing says that $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$ depends on behaviors associated to each participant of G; it also specifies that $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$ does not offer any behaviors of its own.

The following definition relates binary session types and local types: the main difference is that the former do not mention participants. Below, B ranges over base types ($\mathsf{bool}, \mathsf{nat}, \ldots$).

▶ **Definition 4.10** (Local Types→Binary Types). *Mapping* $\langle\!\langle\cdot\rangle\!\rangle$ *from local types* T *(Def. 4.1) into binary types* A *(Def. 3.1) is inductively defined as* $\langle\!\langle \mathsf{end}\rangle\!\rangle = \langle\!\langle B\rangle\!\rangle = \mathbf{1}$ *and*

$$\begin{array}{lcllcl}
\langle\!\langle \mathtt{p}!\{l_i\langle U_i\rangle.T_i\}_{i\in I}\rangle\!\rangle & = & \oplus\{l_i : \langle\!\langle U_i\rangle\!\rangle \otimes \langle\!\langle T_i\rangle\!\rangle\}_{i\in I} & \langle\!\langle \forall\alpha.T\rangle\!\rangle & = & \forall\alpha.\langle\!\langle T\rangle\!\rangle \\
\langle\!\langle \mathtt{p}?\{l_i\langle U_i\rangle.T_i\}_{i\in I}\rangle\!\rangle & = & \&\{l_i : \langle\!\langle U_i\rangle\!\rangle \multimap \langle\!\langle T_i\rangle\!\rangle\}_{i\in I} & \langle\!\langle \exists\alpha.T\rangle\!\rangle & = & \exists\alpha.\langle\!\langle T\rangle\!\rangle \\
\langle\!\langle @_\omega T\rangle\!\rangle & = & @_\omega\langle\!\langle T\rangle\!\rangle & \langle\!\langle {\downarrow}\alpha.T\rangle\!\rangle & = & {\downarrow}\alpha.\langle\!\langle T\rangle\!\rangle
\end{array}$$

Our first characterization result ensures that well-formedness of a global type G guarantees the typability of its medium $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$ using binary session types. Hence, it ensures that multiparty protocols can be analyzed by composing the medium with independently obtained, well-typed implementations for each protocol participant. Crucially, the resulting well-typed process will inherit all correctness properties ensured by binary typability established in § 3.

▶ **Theorem 4.11** (Global Types → Typed Mediums). *If* G *is WF with* $\mathsf{part}(G) = \{\mathtt{p}_1, \ldots, \mathtt{p}_n\}$ *then* $\Omega; \Gamma; c_{\mathtt{p}_1}{:}\langle\!\langle G\!\upharpoonright\!\mathtt{p}_1\rangle\!\rangle[\omega_1], \ldots, c_{\mathtt{p}_n}{:}\langle\!\langle G\!\upharpoonright\!\mathtt{p}_n\rangle\!\rangle[\omega_n] \vdash \mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) :: z : \mathbf{1}[\omega_m]$ *is a compositional typing, for some* Ω, Γ, *with* $\tilde{\omega} = \omega_1, \ldots, \omega_n$. *We assume that* $\omega_i \prec \omega_m$ *for all* $i \in \{1, \ldots, n\}$ *(the medium's domain is accessible by all), and that* $i \neq j$ *implies* $\omega_i \neq \omega_j$.

The second characterization result, given next, is the converse of Theorem 4.11: binary typability precisely delineates the interactions that underlie well-formed multiparty protocols. We need an auxiliary relation on local types, written $\preceq^{\sqcup}_{\downarrow}$, that relates types with branching and "here" type operators, which have silent process interpretations (cf. Figure 1 and [9]). First, we have $T_1 \preceq^{\sqcup}_{\downarrow} T_2$ if there is a T' such that $T_1 \sqcup T' = T_2$ (cf. Def. 4.3). Second, we have $T_1 \preceq^{\sqcup}_{\downarrow} T_2$ if (i) $T_1 = T'$ and $T_2 = {\downarrow}\alpha.T'$ and α does not occur in T'; but also if (ii) $T_1 = {\downarrow}\alpha.T'$ and $T_2 = T'\{^{\omega}/\alpha\}$. (See [9] for a formal definition of $\preceq^{\sqcup}_{\downarrow}$).

▶ **Theorem 4.12** (Well-Typed Mediums → Global Types). *Let* G *be a global type (cf. Def. 4.1). If* $\Omega; \Gamma; c_{\mathtt{p}_1}{:}A_1[\omega_1], \ldots, c_{\mathtt{p}_n}{:}A_n[\omega_n] \vdash \mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c}) :: z : \mathbf{1}[\omega_m]$ *is a compositional typing then* $\exists T_1, \ldots, T_n$ *such that* $G\!\upharpoonright\!\mathtt{p}_j \preceq^{\sqcup}_{\downarrow} T_j$ *and* $\langle\!\langle T_j\rangle\!\rangle = A_j$*, for all* $\mathtt{p}_j \in \mathsf{part}(G)$.

The above theorems offer a *static guarantee* that connects multiparty protocols and well-typed processes. They can be used to establish also *dynamic guarantees* relating the behavior of a global type G and that of its associated set of *multiparty systems* (i.e., the typeful composition of $\mathsf{M}^{\tilde{\omega}}[\![G]\!](\tilde{c})$ with processes for each of $\mathtt{p}_i \in \mathsf{part}(G)$). These dynamic guarantees can be easily obtained by combining Theorems 4.11 and 4.12 with the approach in [7].

5 Related Work

There is a rich history of works on the logical foundations of concurrency (see, e.g., [4, 27, 1, 3]), which has been extended to session-based concurrency by Wadler [47], Dal Lago and Di Giamberardino [35], and others. Medium-based analyses of multiparty sessions were developed in [7] and used in an account of multiparty sessions in an extended classical linear logic [14].

Two salient calculi with distributed features are the Ambient calculus [16], in which processes move across *ambients* (abstractions of administrative domains), and the *distributed π-calculus* (Dpi) [29], which extends the π-calculus with flat locations, local communication, and process migration. While domains in our model may be read as locations, this is just one specific interpretation; they admit various alternative readings (e.g., administrative domains, security-related levels), leveraging the partial view of the domain hierarchy. Type systems for Ambient calculi such as [15, 6] enforce security and communication-oriented properties in terms of ambient movement but do not cover issues of structured interaction, central in our work. Garralda et al. [25] integrate binary sessions in an Ambient calculus, ensuring that session protocols are undisturbed by ambient mobility. In contrast, our type system ensures that both migration and communication are safe and, for the first time in such a setting, satisfy global progress (i.e., session protocols never jeopardize migration and vice-versa).

The multiparty sessions with nested protocols of Demangeon and Honda [19] include a nesting construct that is similar to our new global type $\mathsf{p}\,\mathsf{moves}\,\widetilde{\mathsf{q}}\,\mathsf{to}\,w\,\mathsf{for}\,G_1\,;\,G_2$, which also introduces nesting. The focus in [19] is on modularity in choreographic programming; domains nor domain migration are not addressed. The nested protocols in [19] can have *local* participants and may be parameterized on data from previous actions. We conjecture that our approach can accommodate local participants in a similar way. Data parameterization can be transposed to our logical setting via dependent session types [44, 46]. Asynchrony and recursive behaviors can also be integrated by exploiting existing logical foundations [23, 45].

Balzer et al. [2] overlay a notion of world and accessibility on a system of *shared* session types to ensure deadlock-freedom. Their work differs substantially from ours: they instantiate accessibility as a partial-order, equip sessions with multiple worlds and are not conservative wrt linear logic, being closer to partial-order-based typings for deadlock-freedom [34, 37].

6 Concluding Remarks

We developed a Curry-Howard interpretation of hybrid linear logic as domain-aware session types. Present in processes and types, domain-awareness can account for scenarios where domain information is only determined at runtime. The resulting type system features strong correctness properties for well-typed processes (session fidelity, global progress, termination). Moreover, by leveraging a *parametric* accessibility relation, it rules out processes that communicate with inaccessible domains, thus going beyond the scope of previous works.

As an application of our framework, we presented the first systematic study of domain-awareness in a *multiparty* setting, considering multiparty sessions with domain-aware migration and communication whose semantics is given by a typed (binary) medium process that orchestrates the multiparty protocol. Embedded in a fully distributed domain structure, our medium is shown to strongly encode domain-aware multiparty sessions; it naturally allows us to transpose the correctness properties of our logical development to the multiparty setting.

Our work opens up interesting avenues for future work. Mediums can be seen as *monitors* that enforce the specification of a domain-aware multiparty session. We plan to investigate contract-enforcing mediums building upon works such as [28, 33, 20], which study runtime monitoring in session-based systems. Our enforcement of communication across accessible domains suggests high-level similarities with information flow analyses in multiparty sessions (cf. [13, 12, 17]), but does not capture the directionality needed to model such analyses outright. It would be insightful to establish the precise relationship with such prior works.

References

1 Samson Abramsky. Computational Interpretations of Linear Logic. *Theor. Comput. Sci.*, 111(1&2):3–57, 1993. doi:10.1016/0304-3975(93)90181-R.

2 Stephanie Balzer, Bernardo Toninho, and Frank Pfenning. Manifest Deadlock-Freedom for Shared Session Types. In *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*, pages 611–639, 2019.

3 Emmanuel Beffara. A Concurrent Model for Linear Logic. *Electr. Notes Theor. Comput. Sci.*, 155:147–168, 2006. doi:10.1016/j.entcs.2005.11.055.

4 Gianluigi Bellin and Philip J. Scott. On the pi-Calculus and Linear Logic. *Theor. Comput. Sci.*, 135(1):11–65, 1994. doi:10.1016/0304-3975(94)00104-9.

5 Torben Braüner and Valeria de Paiva. Intuitionistic Hybrid Logic. *J. of App. Log.*, 4:231–255, 2006.

6 Michele Bugliesi and Giuseppe Castagna. Behavioural typing for safe ambients. *Comput. Lang.*, 28(1):61–99, 2002. doi:10.1016/S0096-0551(02)00008-5.

7 Luís Caires and Jorge A. Pérez. Multiparty Session Types Within a Canonical Binary Theory, and Beyond. In *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*, pages 74–95, 2016. Extended version with proofs at https://sites.google.com/a/jorgeaperez.net/www/publications/medium16long.pdf.

8 Luís Caires, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho. Behavioral Polymorphism and Parametricity in Session-Based Communication. In *ESOP*, volume 7792 of *LNCS*. Springer, 2013. doi:10.1007/978-3-642-37036-6_19.

9 Luís Caires, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho. Domain-Aware Session Types (Extended Version). *CoRR*, abs/1907.01318, 2019. arXiv:1907.01318.

10 Luís Caires and Frank Pfenning. Session Types as Intuitionistic Linear Propositions. In *CONCUR*, volume 6269 of *LNCS*, pages 222–236. Springer, 2010. doi:10.1007/978-3-642-15375-4_16.

11 Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Mathematical Structures in Computer Science*, 26(3):367–423, 2016. doi:10.1017/S0960129514000218.

12 Sara Capecchi, Ilaria Castellani, and Mariangiola Dezani-Ciancaglini. Information Flow Safety in Multiparty Sessions. In Bas Luttik and Frank Valencia, editors, *EXPRESS*, volume 64 of *EPTCS*, pages 16–30, 2011. doi:10.4204/EPTCS.64.2.

13 Sara Capecchi, Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Tamara Rezk. Session Types for Access and Information Flow Control. In *CONCUR*, volume 6269 of *LNCS*, pages 237–252. Springer, 2010. doi:10.1007/978-3-642-15375-4_17.

14 Marco Carbone, Sam Lindley, Fabrizio Montesi, Carsten Schürmann, and Philip Wadler. Coherence Generalises Duality: A Logical Explanation of Multiparty Session Types. In *27th International Conference on Concurrency Theory, CONCUR 2016, August 23-26, 2016, Québec City, Canada*, pages 33:1–33:15, 2016.

15 Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Types for the Ambient Calculus. *Inf. Comput.*, 177(2):160–194, 2002. doi:10.1006/inco.2001.3121.

16 Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theor. Comput. Sci.*, 240(1):177–213, 2000. doi:10.1016/S0304-3975(99)00231-5.

17 Ilaria Castellani, Mariangiola Dezani-Ciancaglini, and Jorge A. Pérez. Self-adaptation and secure information flow in multiparty communications. *Formal Asp. Comput.*, 28(4):669–696, 2016. doi:10.1007/s00165-016-0381-3.

18 Kaustuv Chaudhuri, Carlos Olarte, Elaine Pimentel, and Joëlle Despeyroux. Hybrid Linear Logic, revisited. *Mathematical Structures in Computer Science*, 2019. URL: https://hal.inria.fr/hal-01968154.

19 Romain Demangeon and Kohei Honda. Nested Protocols in Session Types. In *CONCUR 2012 - Concurrency Theory - 23rd International Conference, CONCUR 2012, Newcastle upon Tyne, UK, September 4-7, 2012. Proceedings*, pages 272–286, 2012. doi:10.1007/978-3-642-32940-1_20.

20 Romain Demangeon, Kohei Honda, Raymond Hu, Rumyana Neykova, and Nobuko Yoshida. Practical interruptible conversations: distributed dynamic verification with multiparty session types and Python. *Formal Methods in System Design*, 46(3):197–225, 2015. doi:10.1007/s10703-014-0218-8.

21 Pierre-Malo Deniélou and Nobuko Yoshida. Multiparty Compatibility in Communicating Automata: Characterisation and Synthesis of Global Session Types. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 174–186. Springer, 2013. doi:10.1007/978-3-642-39212-2_18.

22 Joëlle Despeyroux, Carlos Olarte, and Elaine Pimentel. Hybrid and Subexponential Linear Logics. *Electr. Notes Theor. Comput. Sci.*, 332:95–111, 2017.

23 Henry DeYoung, Luís Caires, Frank Pfenning, and Bernardo Toninho. Cut Reduction in Linear Logic as Asynchronous Session-Typed Communication. In *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL, CSL 2012, September 3-6, 2012, Fontainebleau, France*, pages 228–242, 2012.

24 Mariangiola Dezani-Ciancaglini and Ugo de'Liguoro. Sessions and Session Types: An Overview. In *WS-FM 2009*, volume 6194 of *LNCS*, pages 1–28. Springer, 2010. doi:10.1007/978-3-642-14458-5_1.

25 Pablo Garralda, Adriana B. Compagnoni, and Mariangiola Dezani-Ciancaglini. BASS: boxed ambients with safe sessions. In Annalisa Bossi and Michael J. Maher, editors, *PPDP*, pages 61–72. ACM, 2006. doi:10.1145/1140335.1140344.

26 Simon J. Gay and Malcolm Hole. Subtyping for session types in the pi calculus. *Acta Inf.*, 42(2-3):191–225, 2005. doi:10.1007/s00236-005-0177-z.

27 Jean-Yves Girard and Yves Lafont. Linear Logic and Lazy Computation. In *TAPSOFT'87: Proceedings of the International Joint Conference on Theory and Practice of Software Development*, pages 52–66, 1987. doi:10.1007/BFb0014972.

28 Hannah Gommerstadt, Limin Jia, and Frank Pfenning. Session-Typed Concurrent Contracts. In *Programming Languages and Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, pages 771–798, 2018.

29 Matthew Hennessy and James Riely. Resource Access Control in Systems of Mobile Agents. *Inf. Comput.*, 173(1):82–120, 2002. doi:10.1006/inco.2001.3089.

30 Kohei Honda. Types for Dynamic Interaction. In *CONCUR*, volume 715 of *LNCS*, pages 509–523. Springer, 1993. doi:10.1007/3-540-57208-2_35.

31 Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *ESOP'98*, LNCS. Springer, 1998. doi:10.1007/BFb0053567.

32 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.

33 Limin Jia, Hannah Gommerstadt, and Frank Pfenning. Monitors and blame assignment for higher-order session types. 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 582–594, 2016.

34 Naoki Kobayashi. A New Type System for Deadlock-Free Processes. In *CONCUR 2006 - Concurrency Theory, 17th International Conference, CONCUR 2006, Bonn, Germany, August 27-30, 2006, Proceedings*, pages 233–247, 2006.

35 Ugo Dal Lago and Paolo Di Giamberardino. Soft Session Types. In *EXPRESS*, volume 64 of *EPTCS*, pages 59–73, 2011. `doi:10.4204/EPTCS.64.5`.

36 Robin Milner, Joachim Parrow, and David Walker. A Calculus of Mobile Processes, part I/II. *Inf. Comput.*, 100(1):1–77, 1992.

37 Luca Padovani. Deadlock and lock freedom in the linear π-calculus. 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 72:1–72:10, 2014.

38 Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear Logical Relations for Session-Based Concurrency. In *ESOP*, volume 7211 of *LNCS*, pages 539–558. Springer, 2012. `doi:10.1007/978-3-642-28869-2_27`.

39 Jorge A. Pérez, Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logical relations and observational equivalences for session-based concurrency. *Inf. Comput.*, 239:254–302, 2014. `doi:10.1016/j.ic.2014.08.001`.

40 Jason Reed. Hybridizing a Logical Framework. *Electr. Notes Theor. Comput. Sci.*, 174(6):135–148, 2007.

41 Davide Sangiorgi. pi-Calculus, Internal Mobility, and Agent-Passing Calculi. *Theor. Comput. Sci.*, 167(1&2):235–274, 1996. `doi:10.1016/0304-3975(96)00075-8`.

42 Davide Sangiorgi and David Walker. *The π-calculus: A Theory of Mobile Processes.* Cambridge University Press, New York, NY, USA, 2001.

43 Alex K. Simpson. *The proof theory and semantics of intuitionistic modal logic.* PhD thesis, University of Edinburgh, UK, 1994. URL: `http://hdl.handle.net/1842/407`.

44 Bernardo Toninho, Luís Caires, and Frank Pfenning. Dependent session types via intuitionistic linear type theory. In *Proc. of PPDP '11*, pages 161–172, New York, NY, USA, 2011. ACM. `doi:10.1145/2003476.2003499`.

45 Bernardo Toninho, Luís Caires, and Frank Pfenning. Corecursion and Non-divergence in Session-Typed Processes. In *Trustworthy Global Computing - 9th International Symposium, TGC 2014, Rome, Italy, September 5-6, 2014. Revised Selected Papers*, pages 159–175, 2014.

46 Bernardo Toninho and Nobuko Yoshida. Depending on Session-Typed Processes. In *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*, pages 128–145, 2018.

47 Philip Wadler. Propositions as sessions. *J. Funct. Program.*, 24(2-3):384–418, 2014. `doi:10.1017/S095679681400001X`.

Reordering Derivatives of Trace Closures of Regular Languages

Hendrik Maarand
Department of Software Science, Tallinn University of Technology, Estonia
hendrik@cs.ioc.ee

Tarmo Uustalu
School of Computer Science, Reykjavik University, Iceland
Department of Software Science, Tallinn University of Technology, Estonia
tarmo@ru.is

Abstract

We provide syntactic derivative-like operations, defined by recursion on regular expressions, in the styles of both Brzozowski and Antimirov, for trace closures of regular languages. Just as the Brzozowski and Antimirov derivative operations for regular languages, these syntactic reordering derivative operations yield deterministic and nondeterministic automata respectively. But trace closures of regular languages are in general not regular, hence these automata cannot generally be finite. Still, as we show, for star-connected expressions, the Antimirov and Brzozowski automata, suitably quotiented, are finite. We also define a refined version of the Antimirov reordering derivative operation where parts-of-derivatives (states of the automaton) are nonempty lists of regular expressions rather than single regular expressions. We define the uniform scattering rank of a language and show that, for a regexp whose language has finite uniform scattering rank, the truncation of the (generally infinite) refined Antimirov automaton, obtained by removing long states, is finite without any quotienting, but still accepts the trace closure. We also show that star-connected languages have finite uniform scattering rank.

2012 ACM Subject Classification Theory of computation → Regular languages; Theory of computation → Concurrency

Keywords and phrases Mazurkiewicz traces, trace closure, regular languages, finite automata, language derivatives, scattering rank, star-connected expressions

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.40

Related Version A full version of this article is available as a preprint [12], `https://arxiv.org/abs/1908.03551`.

Funding *Both authors:* supported by the Estonian Ministry of Education and Research institutional grant no. IUT33-13.
Hendrik Maarand: supported by the ERDF funded Estonian national CoE project EXCITE (2014-2020.4.01.15-0018).

Acknowledgements We thank Pierre-Louis Curien, Jacques Sakarovitch, Simon Doherty, Georg Struth and Ralf Hinze for inspiring discussions, and our anonymous reviewers for the exceptionally thorough and constructive feedback they gave us.

1 Introduction

Traces were introduced to concurrency theory by Mazurkiewicz [13, 14] as an alternative to words. A word can be seen as a linear order that is labelled with letters of the alphabet. Intuitively, the main idea of traces is that the linear order, corresponding to sequentiality, is replaced with a partial order. Sets of words (or word languages) can be used to describe the behaviour of concurrent systems. Similarly, sets of traces (or trace languages) can also be used for this purpose. The difference is that descriptions in terms of traces do not distinguish

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 40; pp. 40:1–40:16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

between different linear extensions (words) of the same partial order (trace) – they are considered equivalent. Different linear extensions of the same partial order can be seen as different observations of the same behaviour.

Given a word language L and a letter a, the derivative of L along a is the language consisting of all the words v such that av belongs to L. An essential difference between words and traces is that a nonempty word (a linear order) has its first letter as the unique minimal element, but a nonempty trace (a partial order) may have several minimal elements. A trace from a trace language can be derived along any of its minimal letters. Clearly, a minimal letter of a trace need not be the first letter of a word representing this trace.

It is well-known that the derivative of a regular word language along a letter is again regular. Brzozowski [5] showed that a regexp for it can be computed from a regexp for the given language, and Antimirov [2] then further optimized this result. We show that these syntactic derivative operations generalize to trace closures (i.e., closures under equivalence) of regular word languages in the form of syntactic *reordering derivative* operations.

The syntactic derivative operations for regular word languages provide ways to construct automata from a regexp. The Brzozowski derivative operation is a function on regexps while the Antimirov derivative operation is a relation. Accordingly, they yield deterministic and nondeterministic automata. The set of Brzozowski derivatives of a regexp (modulo appropriate equations) and the set of Antimirov parts-of-derivatives are finite, hence so are the resulting automata. Our generalizations to trace closures of regular languages similarly give deterministic and nondeterministic automata, but these cannot be finite in general. Still, as we show, for a star-connected expression, the Antimirov and Brzozowski automata, suitably quotiented, are finite. We also develop a finer version of the Antimirov reordering derivative, where parts-of-derivatives are nonempty lists of regexps rather than single regexps, and we show that the set of expressions that can appear in these lists for a given initial regexp is finite. We introduce a new notion of *uniform scattering rank* of a language (a variant of Hashiguchi's scattering rank [7]) and show that, for a regexp whose language has finite uniform rank, a truncation of the refined reordering Antimirov automaton accepts its trace closure despite the removed states, and is finite, without any quotienting.

A full version of this article, with proofs and background material on classical language derivatives and trace closures of regular languages is available as a preprint [12].

2 Preliminaries on Word Languages

An *alphabet* Σ is a finite set (of *letters*). A *word* over Σ is a finite sequence of letters. The set Σ^* of all words over Σ is the free monoid on Σ with the empty word ε as the unit and concatenation of words (denoted by $\cdot$ that can be omitted) as the multiplication. We write $|u|$ for the length of a word u and also $|X|$ for the size of a subalphabet X. By $|u|_a$ we mean the number of occurrences of a in u. By $\Sigma(u)$ we denote the set of letters that appear in u.

A *(word) language* is a subset of Σ^*. The empty word and concatenation of words lift to word languages via $\mathbf{1} =_{\mathrm{df}} \{\varepsilon\}$ and $L \cdot L' =_{\mathrm{df}} \{uv \mid u \in L \wedge v \in L'\}$.

2.1 Derivatives of a Language

A word language L is said to be *nullable* $L{\downarrow}$, if $\varepsilon \in L$. The *derivative* (or *left quotient*)[1] of L along a word u is defined by $D_u L =_{\mathrm{df}} \{v \mid uv \in L\}$. For any L, we have $D_\varepsilon L = L$ as well as $D_{uv} L = D_v(D_u L)$ for any $u, v \in \Sigma^*$, i.e., the operation $D : \mathcal{P}\Sigma^* \times \Sigma^* \to \mathcal{P}\Sigma^*$ is a right action of Σ^* on $\mathcal{P}\Sigma^*$. We also have $L = \{\varepsilon \mid L{\downarrow}\} \cup \bigcup\{\{a\} \cdot D_a L \mid a \in \Sigma\}$, and for any $u \in \Sigma^*$, we have $u \in L$ iff $(D_u L){\downarrow}$.

[1] We use the word "derivative" both for languages and expressions, reserving the word "quotient" for quotients of sets by equivalence relations.

2.2 Regular Languages

The set RE of *regular expressions* (in short, *regexps*) over Σ is given by the grammar $E, F ::= a \mid 0 \mid E + F \mid 1 \mid EF \mid E^*$ where a ranges over Σ.

The *word-language semantics* of regular expressions is given by a function $[\![\_]\!] : \mathsf{RE} \to \mathcal{P}\Sigma^*$ defined recursively by

$$\begin{array}{rcl@{\qquad\qquad}rcl}
[\![a]\!] & =_{\mathrm{df}} & \{a\} & [\![1]\!] & =_{\mathrm{df}} & \mathbf{1} \\
[\![0]\!] & =_{\mathrm{df}} & \emptyset & [\![EF]\!] & =_{\mathrm{df}} & [\![E]\!] \cdot [\![F]\!] \\
[\![E+F]\!] & =_{\mathrm{df}} & [\![E]\!] \cup [\![F]\!] & [\![E^*]\!] & =_{\mathrm{df}} & \mu X.\, \mathbf{1} \cup [\![E]\!] \cdot X
\end{array}$$

A word language L is said to be *regular* (or *rational*) if $L = [\![E]\!]$ for some regexp E. Kleene algebras are defined by an equational theory. It was shown by Kozen [11] that the set $\{[\![E]\!] \mid E \in \mathsf{RE}\}$ of all regular languages together with the language operations $\emptyset$, $\cup$, $\mathbf{1}$, $\cdot$, $(\_)^*$ is the free Kleene algebra on Σ. An important property for us is that $E \doteq F$ iff $[\![E]\!] = [\![F]\!]$ where $\doteq$ refers to valid equations in the Kleene algebra theory.

Kleene's theorem [9] says that a word language is rational iff it is *recognizable*, i.e., accepted by a finite deterministic automaton (acceptance by a finite nondeterministic automaton is an equivalent condition because of determinizability [17]).

2.3 Brzozowski and Antimirov Derivatives

Derivatives of regular languages are regular. A remarkable fact is that they can be computed syntactically, on the level of regular expressions. There are two constructions for this, due to Brzozowski [5] and Antimirov [2]. The Brzozowski and Antimirov derivative operations yield deterministic resp. nondeterministic automata accepting the language of a regular expression E. The Antimirov automaton is finite. The Brzozowski automaton becomes finite when quotiented by associativity, commutativity and idempotence for $+$. Identified up to the Kleene algebra theory, the states of the Brzozowski automaton correspond to the derivatives of the language $[\![E]\!]$. Regular languages can be characterized as languages with finitely many derivatives.

3 Trace Closures of Regular Languages

3.1 Trace Closure of a Word Language

An *independence alphabet* is an alphabet Σ together with an irreflexive and symmetric relation $I \subseteq \Sigma \times \Sigma$ called the *independence* relation. The complement D of I, which is reflexive and symmetric, is called *dependence*. We extend independence to words by saying that two words u and v are independent, uIv, if aIb for all a, b such that $a \in \Sigma(u)$ and $b \in \Sigma(v)$.

Let $\sim^I \subseteq \Sigma^* \times \Sigma^*$ be the least congruence relation on the free monoid Σ^* such that aIb implies $ab \sim^I ba$ for all $a, b \in \Sigma$. If uIv, then $uv \sim^I vu$.

A *(Mazurkiewicz) trace* is an equivalence class of words wrt. $\sim^I$. The equivalence class of a word w is denoted by $[w]^I$.

A word $a_1 \ldots a_n$ where $a_i \in \Sigma$ yields a directed node-labelled acyclic graph as follows. Take the vertex set to be $V =_{\mathrm{df}} \{1, \ldots, n\}$ and label vertex i with a_i. Take the edge set to be $E =_{\mathrm{df}} \{(i, j) \mid i < j \wedge a_i D a_j\}$. This graph (V, E) for a word w is called the *dependence graph* of w and is denoted by $\langle w \rangle_D$. If $w \sim^I z$, then the dependence graphs of w and z are isomorphic, i.e., traces can be identified with dependence graphs up to isomorphism.

The set $\Sigma^*/{\sim^I}$ of all traces is the free partially commutative monoid on (Σ, I). If $I = \emptyset$, then $\Sigma^*/{\sim^I} \cong \Sigma^*$, the set of words, i.e., we recover the free monoid. If $I = \{(a, b) \mid a \neq b\}$, then $\Sigma^*/{\sim^I} \cong \mathcal{M}_{\mathrm{f}}(\Sigma)$, the set of finite multisets over Σ, i.e., the free commutative monoid.

A *trace language* is a subset of $\Sigma^*/{\sim^I}$. Trace languages are in bijection with word languages that are *(trace) closed* in the sense that, if $z \in L$ and $w \sim^I z$, then also $w \in L$. If T is a trace language, then its flattening $L =_{\mathrm{df}} \bigcup T$ is a closed word language. On the other hand, the trace language corresponding to a closed word language L is $T =_{\mathrm{df}} \{t \in \Sigma^*/{\sim^I} \mid \exists z \in t.\, z \in L\} = \{t \in \Sigma^*/{\sim^I} \mid \forall z \in t.\, z \in L\}$.

Given a general (not necessarily closed) word language L, we define its *(trace) closure* $[L]^I$ as the least closed word language that contains L. Clearly $[L]^I = \{w \in \Sigma^* \mid \exists z \in L.\, w \sim^I z\}$ and also $[L]^I = \bigcup\{t \in \Sigma^*/{\sim^I} \mid \exists z \in t.\, z \in L\}$. For any L, we have $[[L]^I]^I = [L]^I$, so $[\_]^I$ is a closure operator. Note also that L is closed iff $[L]^I = L$.

As seen in Section 2.1, the derivative of a word language is the set of all suffixes for a prefix. We now look at what the prefixes and suffixes of a word as a representative of a trace should be. For a word vuv' such that vIu, we can consider u to be its prefix, up to reordering, and vv' to be the suffix. This is because an equivalent word uvv' strictly has u as a prefix and vv' as the suffix. Similarly, we may also want to consider u' to be a prefix of vuv' when $u' \sim^I u$ since $u'vv' \sim^I uvv' \sim^I vuv'$. Note that if a is such a prefix of z, then, by irreflexivity of I, this a is the first a of z. In general, when u is a prefix of z, then the letter occurrences in u uniquely map to letter occurrences in z. We scale these ideas to allow u to be scattered in z as $z = v_0u_1v_1 \ldots u_nv_n$ in either the sense that $u = u_1 \ldots u_n$ or $u \sim^I u_1 \ldots u_n$. We also define bounded versions of scattering that become relevant in Section 5.

▶ **Definition 1.** *For all* $u_1, \ldots, u_n \in \Sigma^+, v_0 \in \Sigma^*, v_1 \ldots, v_{n-1} \in \Sigma^+, v_n \in \Sigma^*, z \in \Sigma^*$,
$u_1, \ldots, u_n \lhd z \rhd v_0, \ldots, v_n =_{\mathrm{df}} z = v_0u_1v_1 \ldots u_nv_n \wedge \forall i.\, \forall j < i.\, v_jIu_i$.

▶ **Definition 2.** *For all* $u, v, z \in \Sigma^*$,

1. $u \lhd z \rhd v =_{\mathrm{df}} \exists n \in \mathbb{N}, u_1, \ldots, u_n, v_0, \ldots, v_n.\, u = u_1 \ldots u_n \wedge v = v_0 \ldots v_n \wedge$
   $u_1, \ldots, u_n \lhd z \rhd v_0, \ldots, v_n$;
2. $u \sim\!\lhd z \rhd v =_{\mathrm{df}} \exists u'.\, u \sim^I u' \wedge u' \lhd z \rhd v$;
3. $u \sim\!\lhd z \rhd\!\sim v =_{\mathrm{df}} \exists u', v'.\, u \sim^I u' \wedge u' \lhd z \rhd v' \wedge v' \sim^I v$.

▶ **Lemma 3.** *For any* $u, v, z \in \Sigma^*$,
$u \lhd z \rhd v \iff \exists! n \in \mathbb{N}, u_1, \ldots, u_n, v_0, \ldots, v_n.\, u = u_1 \ldots u_n \wedge v = v_0 \ldots v_n \wedge$
$u_1, \ldots, u_n \lhd z \rhd v_0, \ldots, v_n$.

▶ **Definition 4.** *For all* $u, v, z \in \Sigma^*$ *and* $N \in \mathbb{N}$,

1. $u \lhd_N z \rhd v =_{\mathrm{df}} \exists n \leq N, u_1, \ldots, u_n, v_0, \ldots, v_n.\, u_1, \ldots, u_n \lhd z \rhd v_0, \ldots, v_n$;
2. *(and* $u \sim\!\lhd_N z \rhd v$ *and* $u \sim\!\lhd_N z \rhd\!\sim v$ *are defined analogously).*

▶ **Example 5.** Let $\Sigma =_{\mathrm{df}} \{a, b, c\}$ and aIb and aIc. Take $z =_{\mathrm{df}} aabcba$. We have $ab \lhd z \rhd acba$ since $a, b \lhd z \rhd \varepsilon, a, cba$. We can visualize this by underlining the subwords of $u =_{\mathrm{df}} ab$ in $z = \varepsilon\underline{a}a\underline{b}cba$. This scattering is valid because εIa, εIb and aIb: recall that Def. 1 requires all underlined subwords u_i to be independent with all non-underlined subwords v_i to their left in z. Similarly we have $aa, a \lhd z \rhd \varepsilon, bcb, \varepsilon$ because $z = \varepsilon\underline{aa}bcb\underline{a}\varepsilon$, εIaa, εIa and $bcbIa$. Note that neither $aa\underline{b}cb\underline{a}\varepsilon$ nor $aabc\underline{ba}\varepsilon$ satisfies the conditions about independence and thus there is no v such that $ba \lhd z \rhd v$. We do have $ba \sim\!\lhd z \rhd acba$ though, since $ba \sim^I ab$ and $a, b \lhd z \rhd \varepsilon, a, cba$.

▶ **Proposition 6.** *For all* $u, v, z \in \Sigma^*, uv \sim^I z \iff u \sim\!\lhd z \rhd\!\sim v$.

3.2 Trace-Closing Semantics of Regular Expressions

We now define a nonstandard word-language semantics of regexps that directly interprets E as the trace closure $[[\![E]\!]]^I$ of its standard regular word-language denotation $[\![E]\!]$.

We have $[\{a\}]^I = \{a\}$, $[\emptyset]^I = \emptyset$, $[L \cup L']^I = [L]^I \cup [L']^I$ and $[\mathbf{1}]^I = \mathbf{1}$. But for general I, we do not have $[L \cdot L']^I = [L]^I \cdot [L']^I$. For example, for $\Sigma =_{\mathrm{df}} \{a, b\}$ and aIb, we have $[\{a\}]^I = \{a\}$, $[\{b\}]^I = \{b\}$ whereas $[\{ab\}]^I = \{ab, ba\} \neq \{ab\} = [\{a\}]^I \cdot [\{b\}]^I$. Hence we need a different concatenation operation.

▶ **Definition 7.**

1. *The I-*reordering concatenation of words *$\cdot^I : \Sigma^* \times \Sigma^* \to \mathcal{P}\Sigma^*$ is defined by*

$$\begin{array}{rcl} \varepsilon \cdot^I v & =_{\mathrm{df}} & \{v\} \\ u \cdot^I \varepsilon & =_{\mathrm{df}} & \{u\} \\ au \cdot^I bv & =_{\mathrm{df}} & \{a\} \cdot (u \cdot^I bv) \cup \{b \mid auIb\} \cdot (au \cdot^I v) \end{array}$$

2. *The lifting of I-reordering concatenation to* languages *is defined by*

$$L \cdot^I L' =_{\mathrm{df}} \bigcup \{u \cdot^I v \mid u \in L \wedge v \in L'\}$$

Note that $\{b \mid auIb\}$ acts as a test: it is either $\emptyset$ or $\{b\}$.

▶ **Example 8.** Let $\Sigma =_{\mathrm{df}} \{a, b\}$ and aIb. Then $a \cdot^I b = \{ab, ba\}$, $aa \cdot^I b = \{aab, aba, baa\}$, $a \cdot^I bb = \{abb, bab, bba\}$ and $ab \cdot^I ba = \{abba\}$. The last example shows that although I-reordering concatenation is defined quite similarly to shuffle, it is different.

▶ **Proposition 9.** *For any $u, v, z \in \Sigma^*$, $z \in u \cdot^I v \iff u \lhd z \rhd v$.*

▶ **Proposition 10.** *For any languages L and L', $[L \cdot L']^I = [L]^I \cdot^I [L']^I$.*

Evidently, if $I = \emptyset$, then reordering concatenation is just ordinary concatenation: $u \cdot^\emptyset v = \{uv\}$. For $I = \Sigma \times \Sigma$, which is forbidden in independence alphabets, as I is required to be irreflexive, it is shuffle: $u \cdot^{\Sigma \times \Sigma} v = u \sqcup\!\sqcup v$. For general I, it has properties similar to concatenation. In particular, we have

$$\begin{array}{rclrcl} \mathbf{1} \cdot^I L & = & L & \emptyset \cdot^I L & = & \emptyset \\ L \cdot^I \mathbf{1} & = & L & (L_1 \cup L_2) \cdot^I L & = & L_1 \cdot^I L \cup L_2 \cdot^I L \\ (L \cdot^I L') \cdot^I L'' & = & L \cdot^I (L' \cdot^I L'') & (L_1 \sqcup\!\sqcup L_2) \cdot^I (L_1' \sqcup\!\sqcup L_2') & \subseteq & (L_1 \cdot^I L_1') \sqcup\!\sqcup (L_2 \cdot^I L_2') \end{array}$$

but also other equations of the concurrent Kleene algebra theory introduced in [8].

We are ready to introduce the closing semantics of regular expressions.

▶ **Definition 11.** *The* trace-closing semantics $[\![\_]\!]^I : \mathsf{RE} \to \mathcal{P}\Sigma^*$ *of regular expressions is defined recursively by*

$$\begin{array}{rclrcl} [\![a]\!]^I & =_{\mathrm{df}} & \{a\} & [\![1]\!]^I & =_{\mathrm{df}} & \mathbf{1} \\ [\![0]\!]^I & =_{\mathrm{df}} & \emptyset & [\![EF]\!]^I & =_{\mathrm{df}} & [\![E]\!]^I \cdot^I [\![F]\!]^I \\ [\![E+F]\!]^I & =_{\mathrm{df}} & [\![E]\!]^I \cup [\![F]\!]^I & [\![E^*]\!]^I & =_{\mathrm{df}} & \mu X.\, \mathbf{1} \cup [\![E]\!]^I \cdot^I X \end{array}$$

Compared to the standard semantics of regular expressions, the difference is in the handling of the EF case (and consequently also the E^* case) due to the cross-commutation that happens in concatenation of traces and must be accounted for by $\cdot^I$.

With $I = \emptyset$, we fall back to the standard interpretation of regular expressions: $[\![E]\!]^\emptyset = [\![E]\!]$. For I a general independence relation, we obtain the desired property that the semantics delivers the trace closure of the language of the regexp.

▶ **Proposition 12.** *For any E, $[\![E]\!]^I$ is trace closed; moreover, $[\![E]\!]^I = [[\![E]\!]]^I$.*

3.3 Properties of Trace Closures of Regular Languages

Trace closures of regular languages are theoretically interesting, as they have intricate properties. For a thorough survey, see Ochmański's handbook chapter [16].

The most important property for us is that the trace closure of a regular language is not necessarily regular. Consider $\Sigma =_{\mathrm{df}} \{a, b\}$, aIb. Let $L =_{\mathrm{df}} [\![(ab)^*]\!]$. The language $[L]^I = \{u \mid |u|_a = |u|_b\}$ is not regular.

The class of trace closures of regular languages over an independence alphabet behaves quite differently from the class of regular languages over an alphabet. For example, the class of trace closures of regular languages over (Σ, I) is closed under complement iff I is quasi-transitive (i.e., its reflexive closure is transitive) [3, 1, 18] (cf. [16, Thm. 6.2.5]); the question of whether the trace closure of the language of a regexp over (Σ, I) is regular is decidable iff I is quasi-transitive [19] (cf. [16, Thm. 6.2.7]).

A closed language is regular iff the corresponding trace language is accepted by a finite asynchronous (a.k.a. Zielonka) automaton [21, 22]. In Section 4.4, we will see further characterizations of regular closed languages based on star-connected expressions.

4 Reordering Derivatives

We are now ready to generalize the Brzozowski and Antimirov constructions for trace closures of regular languages. To this end, we switch to what we call reordering derivatives.

4.1 Reordering Derivative of a Language

Let (Σ, I) be a fixed independence alphabet. We generalize the concepts of (semantic) nullability and derivative of a language to concepts of reorderable part and reordering derivative.

▶ **Definition 13.** *We define the I-*reorderable part *of a language L wrt. a word u by $R^I_u L =_{\mathrm{df}} \{v \in L \mid vIu\}$ and the I-*reordering derivative *along u by $D^I_u L =_{\mathrm{df}} \{v \mid \exists z \in L.\, u \sim\triangleleft\, z \triangleright v\}$.*

By Prop. 9, we can equivalently say that $D^I_u L = \{v \mid \exists z \in L.\, z \in [u]^I \cdot^I v\}$. For a single-letter word a, we get $D^I_a L = \{v_l v_r \mid v_l a v_r \in L \wedge v_l I a\} = \{v \mid \exists z \in L.\, z \in a \cdot^I v\}$. That is, we require some reordering of u (resp. a) to be a prefix, up to reordering, of some word z in L with v as the corresponding strict suffix. (In other words, for the sake of precision and emphasis, we allow reordering of letters within u and across u and v, but not within v.)

▶ **Example 14.** Let $\Sigma =_{\mathrm{df}} \{a, b, c\}$ and aIb. Take $L =_{\mathrm{df}} \{\varepsilon, a, b, ca, aa, bbb, babca, abbaba\}$. We have $R^I_a L = R^I_{aa} L = \{\varepsilon, b, bbb\}$, $D^I_a L = \{\varepsilon, a, bbca, bbaba\}$ and $D^I_{aa} L = \{\varepsilon, bbba\}$.

In the special case $I = \emptyset$, we have $R^{\emptyset}_{\varepsilon} L = L$, $R^{\emptyset}_u L = \{\varepsilon \mid L{\downarrow}\}$ for any $u \neq \varepsilon$, and $D^{\emptyset}_u L = D_u L$. In the general case, the reorderable part and reordering derivative enjoy the following properties.

▶ **Lemma 15.** *For every L,*
1. *$R^I_{\varepsilon} L = L$; for every $u, v \in \Sigma^*$, $R^I_v(R^I_u L) = R^I_{uv} L$;*
2. *for every $u, u' \in \Sigma^*$, $R^I_{\Sigma(u)} L = R^I_{\Sigma(u')} L$.*

We extend R^I to subsets of Σ: by $R^I_X L$, we mean $R^I_u L$ where u is any enumeration of X.

▶ **Lemma 16.** *For every L,*
1. *$D^I_{\varepsilon} L = L$; for any $u, v \in \Sigma^*$, $D^I_v(D^I_u L) = D^I_{uv} L$;*
2. *for any $u, u' \in \Sigma^*$ such that $u \sim^I u'$, we have $D^I_u L = D^I_{u'} L$.*

▶ **Proposition 17.** *For every L,*

1. *for any $u \in \Sigma^*$, $D_u([L]^I) = [D^I_u L]^I$;*
   if L is closed (i.e., $[L]^I = L$), then, for any $u \in \Sigma^*$, $D^I_u L$ is closed and $D_u L = D^I_u L$;
2. *for any $u, v \in \Sigma^*$, $uv \in [L]^I$ iff $v \in [D^I_u L]^I$;*
3. *for any $u \in \Sigma^*$, $u \in [L]^I$ iff $(D^I_u L)\downarrow$;*
4. $[L]^I = \{\varepsilon \mid L\downarrow\} \cup \bigcup_{a\in\Sigma} \{a\} \cdot [D^I_a L]^I$.

▶ **Example 18.** Let $\Sigma =_{\mathrm{df}} \{a, b\}$ and aIb. Take L to be the regular language $[\![(ab)^*]\!]$. We have already noted that the language $[L]^I = \{u \mid |u|_a = |u|_b\}$ is not regular. For any $n \in \mathbb{N}$, $D^I_{b^n} L = \{a^n\} \cdot L = [\![a^n(ab)^*]\!]$ whereas $D_{b^n}([L]^I) = \{a^n\} \cdot^I [L]^I = \{u \mid |u|_a = |u|_b + n\}$. We can see that $[L]^I$ has infinitely many derivatives, none of which are regular, and L has infinitely many reordering derivatives, all regular.

4.2 Brzozowski Reordering Derivative

The reorderable parts and reordering derivatives of regular languages turn out to be regular. We now show that they can be computed syntactically, generalizing the classical syntactic nullability and Brzozowski derivative operations [5].

▶ **Definition 19.** *The I-*reorderable part *and the* Brzozowski I-reordering derivative *of a regexp are given by functions $R^I, D^I : \mathsf{RE} \times \Sigma \to \mathsf{RE}$ and $R^I, D^I : \mathsf{RE} \times \Sigma^* \to \mathsf{RE}$ defined recursively by*

$$
\begin{array}{rcl@{\qquad}rcl}
R^I_a b & =_{\mathrm{df}} & \textsf{if } aIb \textsf{ then } b \textsf{ else } 0 & D^I_a b & =_{\mathrm{df}} & \textsf{if } a = b \textsf{ then } 1 \textsf{ else } 0 \\
R^I_a 0 & =_{\mathrm{df}} & 0 & D^I_a 0 & =_{\mathrm{df}} & 0 \\
R^I_a(E+F) & =_{\mathrm{df}} & R^I_a E + R^I_a F & D^I_a(E+F) & =_{\mathrm{df}} & D^I_a E + D^I_a F \\
R^I_a 1 & =_{\mathrm{df}} & 1 & D^I_a 1 & =_{\mathrm{df}} & 0 \\
R^I_a(EF) & =_{\mathrm{df}} & (R^I_a E)(R^I_a F) & D^I_a(EF) & =_{\mathrm{df}} & (D^I_a E)F + (R^I_a E)(D^I_a F) \\
R^I_a(E^*) & =_{\mathrm{df}} & (R^I_a E)^* & D^I_a(E^*) & =_{\mathrm{df}} & (R^I_a E)^*(D^I_a E)E^* \\
 & & & & & \\
R^I_\varepsilon E & =_{\mathrm{df}} & E & D^I_\varepsilon E & =_{\mathrm{df}} & E \\
R^I_{ua} E & =_{\mathrm{df}} & R^I_a(R^I_u E) & D^I_{ua} E & =_{\mathrm{df}} & D^I_a(D^I_u E)
\end{array}
$$

The regexp $R_u E$ is nothing but E with all occurrences of letters dependent with u replaced with 0. The definition of D is more interesting. Compared to the classical Brzozowski derivative, the nullability condition $E\downarrow$ in the EF case has been replaced with concatenation with the reorderable part $R^I_a E$, and the E^* case has also been adjusted.

The functions R and D on regexps compute their semantic counterparts on the corresponding regular languages.

▶ **Proposition 20.** *For any E,*

1. *for any $a \in \Sigma$, $R^I_a[\![E]\!] = [\![R^I_a E]\!]$ and $D^I_a[\![E]\!] = [\![D^I_a E]\!]$;*
2. *for any $u \in \Sigma^*$, $R^I_u[\![E]\!] = [\![R^I_u E]\!]$ and $D^I_u[\![E]\!] = [\![D^I_u E]\!]$.*

▶ **Proposition 21.** *For any E,*

1. *for any $a \in \Sigma$, $v \in \Sigma^*$, $av \in [\![E]\!]^I \iff v \in [\![D^I_a E]\!]^I$;*
2. *for any $u, v \in \Sigma^*$, $uv \in [\![E]\!]^I \iff v \in [\![D^I_u E]\!]^I$;*
3. *for any $u \in \Sigma^*$, $u \in [\![E]\!]^I \iff (D^I_u E)\downarrow$.*

▶ **Example 22.** Let $\Sigma =_{\mathrm{df}} \{a, b\}$, aIb and $E =_{\mathrm{df}} aa + ab + b$.

$$\begin{array}{rcl} D^I_b E & = & D^I_b aa + D^I_b ab + D^I_b b \\ & = & ((D^I_b a)a + (R^I_b a)(D^I_b a)) + ((D^I_b a)b + (R^I_b a)(D^I_b b)) + D^I_b b \\ & = & (0a + a0) + (0b + a1) + 1 \doteq a + 1 \end{array}$$

$$\begin{array}{rcl} D^I_b(E^*) & = & (R^I_b E)^*(D^I_b E)E^* \\ & = & (aa + a0 + 0)^*((0a + a0) + (0b + a1) + 1)E^* \doteq (aa)^*(a + 1)E^* \end{array}$$

$$D^I_{bb}(E^*) \doteq D^I_b((aa)^*(a + 1)E^*) \doteq (aa)^*(a + 1)(aa)^*(a + 1)E^*$$

As with the classical Brzozowski derivative, we can use the reordering Brzozowski derivative to construct deterministic automata. For a regexp E, take $Q^E =_{\mathrm{df}} \{D^I_u E \mid u \in \Sigma^*\}$, $q^E_0 =_{\mathrm{df}} E$, $F^E =_{\mathrm{df}} \{E' \in Q^E \mid E'{\downarrow}\}$, $\delta^E_a E' =_{\mathrm{df}} D^I_a E'$ for $E' \in Q^E$. By Prop. 21, this automaton accepts the closure $[\![E]\!]^I$. But even quotiented by the full Kleene algebra theory, the quotient of Q^E is not necessarily finite, i.e., we may be able to construct infinitely many different languages by taking reordering derivatives. For the regexp from Example 18, we have $D^I_{b^n}((ab)^*) \doteq a^n(ab)^*$, so it has infinitely many Brzozowski reordering derivatives even up to the Kleene algebra theory. This is only to be expected, as the closure $[\![(ab)^*]\!]^I$ is not regular and cannot possibly have an accepting finite automaton.

4.3 Antimirov Reordering Derivative

Like the classical Brzozowski derivative that was optimized by Antimirov [2], the Brzozowski reordering derivative construction can be optimized by switching from functions on regexps to multivalued functions or relations.

▶ **Definition 23.** *The* Antimirov I-reordering parts-of-derivatives *of a regexp along a letter and a word are relations* $\rightarrow^I \subseteq \mathsf{RE} \times \Sigma \times \mathsf{RE}$ *and* $\rightarrow^{I*} \subseteq \mathsf{RE} \times \Sigma^* \times \mathsf{RE}$ *defined inductively by*

$$\frac{}{a \rightarrow^I (a, 1)} \qquad \frac{E \rightarrow^I (a, E')}{E + F \rightarrow^I (a, E')} \qquad \frac{F \rightarrow^I (a, F')}{E + F \rightarrow^I (a, F')}$$

$$\frac{E \rightarrow^I (a, E')}{EF \rightarrow^I (a, E'F)} \qquad \frac{F \rightarrow^I (a, F')}{EF \rightarrow^I (a, (R^I_a E)F')} \qquad \frac{E \rightarrow^I (a, E')}{E^* \rightarrow^I (a, (R^I_a E)^* E' E^*)}$$

$$\frac{}{E \rightarrow^{I*} (\varepsilon, E)} \qquad \frac{E \rightarrow^{I*} (u, E') \quad E' \rightarrow^I (a, E'')}{E \rightarrow^{I*} (ua, E'')}$$

Here R^I is defined as before. Similarly to the Brzozowski reordering derivative from the previous subsection, the condition $E{\downarrow}$ in the second EF rule has has been replaced with concatenation with $R^I_a E$, and the E^* rule has been adjusted.

Collectively, the Antimirov reordering parts-of-derivatives of a regexp E compute the semantic reordering derivative of the language $[\![E]\!]$.

▶ **Proposition 24.** *For any E,*

1. *for any $a \in \Sigma$, $D^I_a [\![E]\!] = \bigcup\{[\![E']\!] \mid E \rightarrow^I (a, E')\}$;*
2. *for any $u \in \Sigma^*$, $D^I_u [\![E]\!] = \bigcup\{[\![E']\!] \mid E \rightarrow^{I*} (u, E')\}$.*

▶ **Proposition 25.** *For any E,*

1. *for any $a \in \Sigma$, $v \in \Sigma^*$, $av \in [\![E]\!]^I \iff \exists E'.\, E \rightarrow^I (a, E') \wedge v \in [\![E']\!]^I$;*
2. *for any $u, v \in \Sigma^*$, $uv \in [\![E]\!]^I \iff \exists E'.\, E \rightarrow^{I*} (u, E') \wedge v \in [\![E']\!]^I$;*
3. *for any $u \in \Sigma^*$, $u \in [\![E]\!]^I \iff \exists E'.\, E \rightarrow^{I*} (u, E') \wedge E'{\downarrow}$.*

▶ **Example 26.** Let us revisit Example 22. The Antimirov reordering parts-of-derivatives of E along b are $a1$ and 1:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{b \to^I (b,1)}}{ab \to^I (b,a1)}}{ab+b \to^I (b,a1)}}{aa+ab+b \to^I (b,a1)} \qquad \dfrac{\dfrac{\dfrac{}{b \to^I (b,1)}}{ab+b \to^I (b,1)}}{aa+ab+b \to^I (b,1)}$$

The Antimirov reordering parts-of-derivatives of E^* along b are therefore $E_b^*(a1)E^*$ and $E_b^* 1 E^*$ where $E_b =_{\mathrm{df}} R_b^I E = aa+a0+0$. Recall that, for the Brzozowski reordering derivative, we computed $D_b^I E = (0a+a0)+(0b+a1)+1$ and $D_b^I E^* = E_b^*((0a+a0)+(0b+a1)+1)E^*$.

Like the classical Antimirov construction, the Antimirov reordering parts-of-derivatives of a regexp E give a nondeterministic automaton by $Q^E =_{\mathrm{df}} \{E' \mid \exists u \in \Sigma^*.\, E \to^{I*} (u, E')\}$, $I^E =_{\mathrm{df}} \{E\}$, $F^E =_{\mathrm{df}} \{E' \in Q^E \mid E'{\downarrow}\}$, $E' \to^E (a, E'') =_{\mathrm{df}} E' \to^I (a, E'')$ for $E', E'' \in Q^E$. This automaton accepts $[\![E]\!]^I$ by Prop. 25, but is generally infinite, also if quotiented by the full Kleene algebra theory. Revisiting Example 18 again, $(ab)^*$ must have infinitely many Antimirov reordering parts-of-derivatives modulo the Kleene algebra theory since $[\![(ab)^*]\!]^I$ is not regular and cannot have a finite accepting nondeterministic automaton. Specifically, it has $(a0)^*((a1)\ldots((a0)^*((a1)(ab)^*))\ldots) \doteq a^n(ab)^*$ as its single reordering part-of-derivative along b^n.

However, if quotienting the Antimirov automaton for E by some sound theory (a theory weaker than the Kleene algebra theory) makes it finite, then the Brzozowski automaton can also be quotiented to become finite.

▶ **Proposition 27.** *For any E,*
1. *for any $a \in \Sigma$, $D_a^I E \doteq \sum\{E' \mid E \to^I (a, E')\}$;*
2. *for any $u \in \Sigma^*$, $D_u^I E \doteq \sum\{E' \mid E \to^{I*} (u, E')\}$*

(using the semilattice equations for $0, +$, that 0 is zero, and distributivity of $\cdot$ over $+$).

▶ **Corollary 28.** *If some quotient of the Antimirov automaton for E (accepting $[\![E]\!]^I$) is finite, then also some quotient of the Brzozowski automaton is finite.*

4.4 Star-Connected Expressions

Star-connected expressions are important as they characterize regular closed languages. A corollary of that is a further characterization of such languages in terms of a "concurrent" semantics of regexps that interprets Kleene star nonstandardly as "concurrent star".

▶ **Definition 29.** *A word $w \in \Sigma^*$ is connected if its dependence graph $\langle w \rangle_D$ is connected. A language $L \subseteq \Sigma^*$ is connected if every word $w \in L$ is connected.*

▶ **Definition 30.**
1. Star-connected expressions *are a subset of the set of all regexps defined inductively by:* 0, 1 *and $a \in \Sigma$ are star-connected. If E and F are star-connected, then so are $E+F$ and EF. If E is star-connected and $[\![E]\!]$ is connected, then E^* is star-connected.*
2. *A language L is said to be* star-connected *if $L = [\![E]\!]$ for some star-connected regexp.*

Ochmański [15] proved that a closed language is regular iff it is the closure of a star-connected language (cf. [16, Thm. 6.3.13]). This means that, for any regexp E, the language $[\![E]\!]^I$ is regular iff there exists a (generally different!) star-connected expression E' such that $[\![E]\!]^I = [\![E']\!]^I$. As a corollary, a closed language is regular iff it is the closure of the concurrent denotation of some regexp (cf. [16, Thm. 6.3.16]).

4.5 Automaton Finiteness for Star-Connected Expressions

We now show that the set of Antimirov reordering parts-of-derivatives of a star-connected expression is finite modulo suitable equations.

▶ **Lemma 31.** *If $[\![E]\!]$ is connected, then, for every $u \in \Sigma^+$ and E' such that $E \to^{I*} (u, E')$, either $R^I_u E' \doteq 0$ or $R^I_u E' \doteq 1$ (using the equations involving 0 and 1 only and that 0 is zero).*

▶ **Lemma 32.** *If $[\![E]\!]$ is connected and $E^* \to^{I*} (u, E')$, then there exists E'' such that $E' \doteq E''$ (using, additionally, the monoid equations for $1, \cdot$ and the equation $F^* \cdot F^* \doteq F^*$) and E'' contains at most $|\Sigma|$ subexpressions of the form $(R^I_X E)^*$ where $\emptyset \subset X \subseteq \Sigma$.*

▶ **Proposition 33.** *If E is star-connected, then a suitable sound quotient of the state set $\{E' \mid \exists u \in \Sigma^*.\ E \to^{I*} (u, E')\}$ of the Antimirov automaton for E (accepting $[\![E]\!]^I$) is finite.*

5 Uniform Scattering Rank of a Language

We proceed to defining the notion of uniform scattering rank of a language and show that star-connected expressions define languages with uniform scattering rank.

5.1 Scattering Rank vs. Uniform Scattering Rank

The notion of scattering rank of a language (a.k.a. distribution rank, k-block testability) was introduced by Hashiguchi [7].

▶ **Definition 34.** *A language L has* (I-scattering) rank *at most N if*
$\forall u, v.\, uv \in [L]^I \implies \exists z \in L.\, u \sim\lhd_N z \rhd\sim v$.

Hashiguchi [7] showed that having rank is a sufficient condition for regularity of the trace closure of a regular language (cf. [16, Prop. 6.3.2]). But it is not a necessary condition: for $\Sigma =_{\mathrm{df}} \{a, b\}$, aIb, the language $L =_{\mathrm{df}} [\![(aa+ab+ba+bb)^*]\!]$ has rank 1 (as does any nontrivial closed language), but it is not star-connected.

We wanted to show that a truncation of the refined Antimirov automaton (which we define in Section 6) is finite for regexps whose language has rank (i.e., the language has rank at most N for some $N \in \mathbb{N}$). But it turns out, as we shall see, that rank does not quite work for this. For this reason, we introduce a stronger notion that we call uniform scattering rank.

▶ **Definition 35.** *A language L has* uniform (I-scattering) rank *at most N if*
$\forall w \in [L]^I.\ \exists z \in L.\ \forall u, v.\, w = uv \implies u \sim\lhd_N z \rhd\sim v$.

The difference between the two definitions is that, in the uniform case, the choice of z depends only on w whereas, in the non-uniform case, it depends on the particular split of w as $w = uv$, i.e., for every such split of w we may choose a different z.

▶ **Lemma 36.** *If L has uniform rank at most N, then L has rank at most N.*

The converse of the above lemma does not hold – there are languages with uniform rank greater than rank. Furthermore, there are languages that have rank but no uniform rank.

▶ **Proposition 37.** *Let $\Sigma =_{\mathrm{df}} \{a, b, c\}$, aIb and $E =_{\mathrm{df}} a^*b^*c(ab)^*(a^*+b^*)+(ab)^*(a^*+b^*)ca^*b^*$. The language $[\![E]\!]$ has rank 2, but no uniform rank.*

5.2 Star-Connected Languages Have Uniform Rank

Klunder et al. [10] established that star-connected languages have rank. We will now show that star-connected languages also have uniform rank, by strengthening their proof.

It can be seen that if L_1 and L_2 have uniform rank at most N_1 and N_2, then $L_1 \cup L_2$ has uniform rank at most $\max(N_1, N_2)$ and $L_1 \cdot L_2$ has uniform rank at most $N_1 + N_2$. If a general L has uniform rank at most N, then L^* need not have uniform rank. For example, for $\Sigma =_{\mathrm{df}} \{a, b\}$, aIb, the language $\{ab\}$ has uniform rank 1, but $\{ab\}^*$ is without rank, so also without uniform rank. But if L is also connected, then L^* has uniform rank at most $(N + 1) \cdot |\Sigma|$.

▶ **Proposition 38.** *If E is star-connected, then the language $[\![E]\!]$ has uniform rank.*

6 Antimirov Reordering Derivative and Uniform Rank

We have seen that the reordering language derivative $D^I_u L$ allows u to be scattered in a word $z \in L$ as $u_1, \ldots, u_n \lhd z \rhd v_0, \ldots, v_n$ where $u \sim^I u_1 \ldots u_n$. We will now consider a version of the Antimirov reordering derivative operation that delivers lists of regexps for the possible $v_0, \ldots, v_n$ rather than just single regexps for their concatenations $v_0 \ldots v_n$.

6.1 Refined Antimirov Reordering Derivative

The refined reordering parts-of-derivative of a regexp E along a letter a are pairs of regexps E_l, E_r. For any word $w = av \in [\![E]\!]^I$, there must be an equivalent word $z = v_l a v_r \in [\![E]\!]$. Instead of describing the words $v_l v_r$ obtainable by removing a minimal occurrence of a in a word $z \in [\![E]\!]$, the refined parts-of-derivative describe the subwords v_l, v_r that were to the left and right of this a in z: it must be the case that $v_l \in [\![E_l]\!]$ and $v_r \in [\![E_r]\!]$ for one of the pairs E_l, E_r. For a longer word u, the refined reordering derivative operation gives lists of regexps $E_0, \ldots, E_n$ fixing what the lists of subwords $v_0, \ldots, v_n$ can be in words $z = v_0 u_1 v_1 \ldots u_n v_n \in [\![E]\!]$ equivalent to a given word $w = uv \in [\![E]\!]^I$.

▶ **Definition 39.** *The* (unbounded and bounded) refined Antimirov I-reordering *parts-of-derivatives of a regexp along a letter and a word are given by relations $\to^I \subseteq \mathsf{RE} \times \Sigma \times \mathsf{RE} \times \mathsf{RE}$, $\Rightarrow^I \subseteq \mathsf{RE}^+ \times \Sigma \times \mathsf{RE}^+$, $\to^{I*} \subseteq \mathsf{RE} \times \Sigma^* \times \mathsf{RE}^+$, $\Rightarrow^I_N \subseteq \mathsf{RE}^{+\leq N+1} \times \Sigma \times \mathsf{RE}^{+\leq N+1}$, and $\to^{I*}_N \subseteq \mathsf{RE} \times \Sigma^* \times \mathsf{RE}^{+\leq N+1}$ defined inductively by*

$$\frac{}{a \to^I (a; 1, 1)} \qquad \frac{E \to^I (a; E_l, E_r)}{E + F \to^I (a; E_l, E_r)} \qquad \frac{F \to^I (a; F_l, F_r)}{E + F \to^I (a; F_l, F_r)}$$

$$\frac{E \to^I (a; E_l, E_r)}{EF \to^I (a; E_l, E_r F)} \qquad \frac{F \to^I (a; F_l, F_r)}{EF \to^I (a; (R^I_a E) F_l, F_r)} \qquad \frac{E \to^I (a; E_l, E_r)}{E^* \to^I (a; (R^I_a E)^* E_l, E_r E^*)}$$

$$\frac{E \to^I (a; E_l, E_r) \quad |\Gamma, \Delta| < N}{\Gamma, E, \Delta \Rightarrow^I_N (a; R^I_a \Gamma, E_l, E_r, \Delta)} \qquad \frac{E \to^I (a; E_l, E_r) \quad E_l{\downarrow} \quad |\Gamma| > 0}{\Gamma, E, \Delta \Rightarrow^I_N (a; R^I_a \Gamma, E_r, \Delta)}$$

$$\frac{E \to^I (a; E_l, E_r) \quad E_r{\downarrow} \quad |\Delta| > 0}{\Gamma, E, \Delta \Rightarrow^I_N (a; R^I_a \Gamma, E_l, \Delta)} \qquad \frac{E \to^I (a; E_l, E_r) \quad E_l{\downarrow} \quad E_r{\downarrow} \quad |\Gamma| > 0 \quad |\Delta| > 0}{\Gamma, E, \Delta \Rightarrow^I_N (a; R^I_a \Gamma, \Delta)}$$

$$\frac{}{E \to^{I*}_N (\varepsilon; E)} \qquad \frac{E \to^{I*}_N (u; \Gamma) \quad \Gamma \Rightarrow^I_N (a; \Gamma')}{E \to^{I*}_N (ua; \Gamma')}$$

By $\mathsf{RE}^{+\leq N+1}$ we mean nonempty lists of regexps of length at most $N + 1$. The relations $\Rightarrow^I$ and $\to^{I*}$ are defined exactly as $\Rightarrow^I_N$ and $\to^{I*}_N$ but with the condition $|\Gamma, \Delta| < N$ of the first rule of $\Rightarrow^I_N$ dropped. The operation R^I_a is extended to lists of regexps in the obvious way.

We have several rules for deriving a list of regexps along a. If E is split into E_l, E_r and neither of them is nullable, then, in the N-bounded case, we require that the given list is shorter than $N+1$ since the new list will be longer by 1. If one of E_l, E_r is nullable, not the first resp. last in the list and we choose to drop it, then the new list will be of the same length. If both are nullable, not the first resp. last and we opt to drop both, then the new list will be shorter by 1. They must be droppable under these conditions to handle the situation when a word z has been split as $v_0 u_1 v_1 \dots u_k v_k u_{k+1} \dots u_n v_n$ and v_k is further being split as $v_l a v_r$ while v_l or v_r is empty. If $k \neq 0$ and v_l is empty, we must join u_k and a into $u_k a$. If $k \neq n$ and v_r is empty, we must join a and u_{k+1} into $a u_{k+1}$. If k is neither 0 nor n and both v_l and v_r are empty, we must join all three of u_k, a and u_{k+1} into $u_k a u_{k+1}$. The length of the new list of regexps is always at least 2.

▶ **Proposition 40.** *For any E,*

1. *for any $a \in \Sigma, v_l, v_r \in \Sigma^*$,*

$$v_l I a \wedge v_l a v_r \in [\![E]\!] \iff \exists E_l, E_r.\, E \to^I (a; E_l, E_r) \wedge v_l \in [\![E_l]\!] \wedge v_r \in [\![E_r]\!];$$

2. *for any $u \in \Sigma^*, n \in \mathbb{N}, v_0 \in \Sigma^*, v_1, \dots, v_{n-1} \in \Sigma^+, v_n \in \Sigma^*$,*

$$\exists z \in \Sigma^*, u_1, \dots, u_n \in \Sigma^+.\, z \in [\![E]\!] \wedge u \sim^I u_1 \dots u_n \wedge u_1, \dots, u_n \lhd z \rhd v_0, \dots, v_n$$
$$\iff$$
$$\exists E_0, \dots, E_n.\, E \to^{I*} (u; E_0, \dots, E_n) \wedge \forall j.\ v_j \in [\![E_j]\!].$$

▶ **Proposition 41.** *For any E,*

1. *for any $a \in \Sigma, v \in \Sigma^*$, the following are equivalent:*
   - **a.** $av \in [\![E]\!]^I$;
   - **b.** $\exists v_l, v_r \in \Sigma^*.\, v \sim^I v_l v_r \wedge v_l I a \wedge v_l a v_r \in [\![E]\!]$;
   - **c.** $\exists v_l, v_r \in \Sigma^*.$
     $v \sim^I v_l v_r \wedge \exists E_l, E_r.\, E \to^I (a; E_l, E_r) \wedge v_l \in [\![E_l]\!] \wedge v_r \in [\![E_r]\!]$;
   - **d.** $\exists v_l, v_r \in \Sigma^*.$
     $v \in v_l \cdot^I v_r \wedge \exists E_l, E_r.\, E \to^I (a; E_l, E_r) \wedge v_l \in [\![E_l]\!]^I \wedge v_r \in [\![E_r]\!]^I$.
2. *for any $u, v \in \Sigma^*$, the following are equivalent:*
   - **a.** $uv \in [\![E]\!]^I$;
   - **b.** $\exists z \in [\![E]\!].\ u \sim\lhd z \rhd\sim v$;
   - **c.** $\exists n \in \mathbb{N}, v_0 \in \Sigma^*, v_1, \dots, v_{n-1} \in \Sigma^+, v_n \in \Sigma^*.\, v \sim^I v_0 v_1 \dots v_n \wedge$
     $\exists E_0, \dots, E_n.\, E \to^{I*} (u; E_0, \dots, E_n) \wedge \forall j.\, v_j \in [\![E_j]\!]$;
   - **d.** $\exists n \in \mathbb{N}, v_0 \in \Sigma^*, v_1, \dots, v_{n-1} \in \Sigma^+, v_n \in \Sigma^*.\, v \in v_0 \cdot^I v_1 \cdot^I \dots \cdot^I v_n \wedge$
     $\exists E_0, \dots, E_n.\, E \to^{I*} (u; E_0, \dots, E_n) \wedge \forall j.\, v_j \in [\![E_j]\!]^I$.
3. *for any $u \in \Sigma^*$,*
   $u \in [\![E]\!]^I \iff (u = \varepsilon \wedge E{\downarrow}) \vee (u \neq \varepsilon \wedge \exists E_0, E_1.\, E \to^{I*} (u; E_0, E_1) \wedge E_0{\downarrow} \wedge E_1{\downarrow})$.

▶ **Corollary 42.** *For any E such that $[\![E]\!]$ has uniform rank at most N,*

1. *for any $u, v \in \Sigma^*$, the following are equivalent:*
   - **a.** $uv \in [\![E]\!]^I$;
   - **b.** $\exists z \in [\![E]\!].\ \forall u', u''.\, u = u'u'' \implies u' \sim\lhd_N z \rhd\sim u''v$;
   - **c.** $\exists n \leq N, v_0 \in \Sigma^*, v_1, \dots, v_{n-1} \in \Sigma^+, v_n \in \Sigma^*.\, v \sim^I v_0 v_1 \dots v_n \wedge$
     $\exists E_0, \dots, E_n.\, E \to^{I*}_N (u; E_0, \dots, E_n) \wedge \forall j.\, v_j \in [\![E_j]\!]$;
   - **d.** $\exists n \leq N, v_0 \in \Sigma^*, v_1, \dots, v_{n-1} \in \Sigma^+, v_n \in \Sigma^*.\, v \in v_0 \cdot^I v_1 \cdot^I \dots \cdot^I v_n \wedge$
     $\exists E_0, \dots, E_n.\, E \to^{I*}_N (u; E_0, \dots, E_n) \wedge \forall j.\, v_j \in [\![E_j]\!]^I$.
2. *for any $u \in \Sigma^*$,*
   $u \in [\![E]\!]^I \iff (u = \varepsilon \wedge E{\downarrow}) \vee (u \neq \varepsilon \wedge \exists E_0, E_1.\, E \to^{I*}_N (u; E_0, E_1) \wedge E_0{\downarrow} \wedge E_1{\downarrow})$.

▶ **Example 43.** We go back to Example 22. Recall that $E =_{\mathrm{df}} aa + ab + b$ and $E_b =_{\mathrm{df}} R^I_b E = aa + a0 + 0$. Here is one of the refined reordering parts-of-derivatives of E^* along bb.

$$\frac{\dfrac{\dfrac{}{E^* \to_2^{I*} (\varepsilon; E^*)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{b \to^I (b; 1, 1)}}{ab \to^I (b; a1, 1)}}{ab + b \to^I (b; a1, 1)}}{aa + ab + b \to^I (b; a1, 1)}}{E^* \to^I (b; E^*_b(a1), 1E^*)} \quad 0 < 2}{E^* \Rightarrow_2^I (b; E^*_b(a1), 1E^*)}}{E^* \to_2^{I*} (b; E^*_b(a1), 1E^*)} \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{b \to^I (b; 1, 1)}}{ab \to^I (b; a1, 1)}}{ab + b \to^I (b; a1, 1)}}{aa + ab + b \to^I (b; a1, 1)}}{E^* \to^I (b; E^*_b(a1), 1E^*)}}{1E^* \to^I (b; 1(E^*_b(a1)), 1E^*)} \quad 1 < 2}{E^*_b(a1), 1E^* \Rightarrow_2^I (b; E^*_b(a1), 1(E^*_b(a1)), 1E^*)}}{E^* \to_2^{I*} (bb; E^*_b(a1), 1(E^*_b(a1)), 1E^*)}$$

In this example, we chose $N =_{\mathrm{df}} 2$. The regexp $1(E^*_b(a1)) \doteq (aa)^*a$ is not nullable, so we could not have dropped it. From here we cannot continue by deriving along a third b by again taking it from the summand ab of E in $1E^*$, as this would produce another nondroppable $1(E^*_b(a1))$ and make the list too long (longer than 3). For example, we are not allowed to establish $w =_{\mathrm{df}} bbbaaa \in [\![E^*]\!]^I$ (by deriving E^* along w and checking if we can arrive at E_0, E_1 with both E_0, E_1 nullable), mandated by $z =_{\mathrm{df}} ababab \in [\![E^*]\!]$, but we are allowed to do so because of $z' =_{\mathrm{df}} bbabaa \in [\![E^*]\!]$. The word z is not useful since among the splits of w as $w = uv$ there is $u =_{\mathrm{df}} bbb$, $v =_{\mathrm{df}} aaa$, which splits z as $u \sim\!\lhd\ z \rhd\!\sim v$ scattering u into 3 blocks as $z = a\underline{b}a\underline{b}a\underline{b}$ (we underline the letters from u); the full sequence of these corresponding splits of z is $ababab$, $a\underline{b}abab$, $a\underline{b}a\underline{b}ab$, $a\underline{b}a\underline{b}a\underline{b}$, $\underline{ab}a\underline{b}a\underline{b}$, $\underline{abab}a\underline{b}$, $\underline{ababab}$. The word z', on the contrary, is fine because, for every split of w as $w = uv$, there are at most two blocks of letters from u in z': $bbabaa$, $\underline{b}babaa$, $\underline{bb}abaa$, $\underline{bb}a\underline{b}aa$, $\underline{bbab}aa$, $\underline{bbaba}a$, $\underline{bbabaa}$. The choice $N = 2$ suffices for accepting all of $[\![E^*]\!]^I$, since $[\![E^*]\!]$ happens to have uniform rank 2.

The refined Antimirov reordering parts-of-derivatives of a regexp E give a nondeterministic automaton by $Q^E =_{\mathrm{df}} \{\Gamma \mid \exists u \in \Sigma^*.\, E \to^{I*} (u; \Gamma)\}$, $I^E =_{\mathrm{df}} \{E\}$, $F^E =_{\mathrm{df}} \{E \mid E\!\downarrow\} \cup \{E_0, E_1 \in Q^E \mid E_0\!\downarrow \wedge E_1\!\downarrow\}$, $\Gamma \to^E (a; \Gamma') =_{\mathrm{df}} \Gamma \Rightarrow^I (a; \Gamma')$ for $\Gamma, \Gamma' \in Q^E$. By Prop. 41, this automaton accepts $[\![E]\!]^I$. It is generally not finite as Q^E can contain states Γ of any length.

Given $N \in \mathbb{N}$, another automaton is obtained by restricting Q^E, F^E and $\to^E$ to $Q^E_N =_{\mathrm{df}} \{\Gamma \mid \exists u \in \Sigma^*.\, E \to^{I*}_N (u; \Gamma)\}$, $F^E_N =_{\mathrm{df}} \{E \mid E\!\downarrow\} \cup \{E_0, E_1 \in Q^E_N \mid E_0\!\downarrow \wedge E_1\!\downarrow\}$, $\Gamma \to^E_N (a; \Gamma') =_{\mathrm{df}} \Gamma \Rightarrow^I_N (a; \Gamma')$ for $\Gamma, \Gamma' \in Q^E_N$. By Cor. 42, if $[\![E]\!]$ has uniform rank at most N, then this smaller automaton accepts $[\![E]\!]^I$ despite the truncation. If $[\![E]\!]$ does not have uniform rank or we choose N smaller than the uniform rank, then the N-truncated automaton recognizes a proper subset of $[\![E]\!]^I$. Prop. 37 gives an example of this: however we choose N, the N-truncated automaton fails to accept the word $a^n b^n c a^n b^n$ for $n > N$. This happens because $[\![E]\!]$ does not have uniform rank (and that it has rank 2 does not help).

6.2 Automaton Finiteness for Regular Expressions with Uniform Rank

Is the N-truncated Antimirov automaton finite? The states Γ of Q^E_N are all of length at most $N + 1$, so there is hope. The automaton will be finite if we can find a finite set containing all the individual regexps E' appearing in the states Γ. We now define such a set $E^{\to *}$.

▶ **Definition 44.** *We define functions* $(\_)^{\rightsquigarrow+}, \mathbf{R}, (\_)^{\rightarrow+}, (\_)^{\rightarrow*} : \mathsf{RE} \to \mathcal{P}\mathsf{RE}$ *by*

$$\begin{array}{rcl@{\qquad}rcl}
a^{\rightsquigarrow+} & =_{\mathrm{df}} & \{1\} & (E+F)^{\rightsquigarrow+} & =_{\mathrm{df}} & E^{\rightsquigarrow+} \cup F^{\rightsquigarrow+} \\
0^{\rightsquigarrow+} & =_{\mathrm{df}} & \emptyset & 1^{\rightsquigarrow+} & =_{\mathrm{df}} & \emptyset
\end{array}$$

$$\begin{array}{rcl}
(EF)^{\rightsquigarrow+} & =_{\mathrm{df}} & E^{\rightsquigarrow+} \cup F^{\rightsquigarrow+} \cup E^{\rightsquigarrow+} \cdot \{F\} \cup \{E\} \cdot F^{\rightsquigarrow+} \cup E^{\rightsquigarrow+} \cdot F^{\rightsquigarrow+} \\
(E^*)^{\rightsquigarrow+} & =_{\mathrm{df}} & E^{\rightsquigarrow+} \cup \{E^*\} \cdot E^{\rightsquigarrow+} \cup E^{\rightsquigarrow+} \cdot \{E^*\} \cup E^{\rightsquigarrow+} \cdot (\{E^*\} \cdot E^{\rightsquigarrow+}) \cup \ (E^{\rightsquigarrow+} \cdot \{E^*\}) \cdot E^{\rightsquigarrow+}
\end{array}$$

$$\begin{array}{rcl}
\mathbf{R}E & =_{\mathrm{df}} & \{R^I_X E \mid X \subseteq \Sigma\} \\
E^{\rightarrow+} & =_{\mathrm{df}} & \mathbf{R}(E^{\rightsquigarrow+}) \\
E^{\rightarrow*} & =_{\mathrm{df}} & \{E\} \cup E^{\rightarrow+}
\end{array}$$

▶ **Proposition 45.**

1. *For any E, the set $E^{\rightarrow*}$ is finite.*
2. *For any E and X, we have $(R^I_X E)^{\rightarrow*} \subseteq R^I_X(E^{\rightarrow*})$.*
3. *For any E, a and E_l, E_r, if $E \rightarrow^I (a; E_l, E_r)$, then $E_l \in R^I_a(E^{\rightsquigarrow+})$ and $E_r \in E^{\rightsquigarrow+}$.*
4. *For any E, E', X, a, E'_l, E'_r, if $E' \in R^I_X(E^{\rightsquigarrow+})$ and $E' \rightarrow^I (a; E'_l, E'_r)$, then $E'_l \in R^I_{Xa}(E^{\rightsquigarrow+})$ and $E'_r \in R^I_X(E^{\rightsquigarrow+})$.*
5. *For any E, u and $E_0, \ldots, E_n$, if $E \rightarrow^{I*} (u; E_0, \ldots, E_n)$, then $\forall j.\, E_j \in E^{\rightarrow*}$.*

▶ **Proposition 46.** *For every E and N, the state set $\{\Gamma \mid \exists u \in \Sigma^*.\, E \rightarrow^{I*}_N (u; \Gamma)\}$ of the N-truncated refined Antimirov automaton for E (accepting $[\![E]\!]^I$ if $[\![E]\!]$ has uniform rank at most N) is finite.*

7 Related Work

Syntactic derivative constructions for regular expressions extended with constructors for (versions of) the shuffle operation have been considered, for example, by Sulzmann and Thiemann [20] for the Brzozowski derivative and by Broda et al. [4] for the Antimirov derivative. This is relevant to our derivatives since $L \cdot^I L'$ is by definition a language between $L \cdot L'$ and $L \sqcup\!\sqcup L'$. Thus our Brzozowski and Antimirov reordering derivatives of EF must be between the classical Brzozowski and Antimirov derivatives of EF and $E \sqcup\!\sqcup F$.

8 Conclusion and Future Work

We have shown that the Brzozowski and Antimirov derivative operations generalize to trace closures of regular languages in the form of reordering derivative operations. The sets of Brzozowski resp. Antimirov reordering (parts-of-)derivatives of a regexp are generally infinite, so the deterministic and nondeterministic automata that they give, accepting the trace closure, are generally infinite. Still, if the regexp is star-connected, their appropriate quotients are finite. Also, the set of N-bounded refined Antimirov reordering parts-of-derivatives is finite without quotienting, and we showed that, if the language of the regexp has uniform rank at most N, the N-truncated refined Antimirov automaton accepts the trace closure. We also proved that star-connected expressions define languages with finite uniform rank.

Our intended application for this is operational semantics in the context of relaxed memory (where, e.g., shadow writes, i.e., writes from local buffers to shared memory, can be reorderable with other actions). For sequential composition EF it is usually required that, to execute any action from F, execution of E must have completed. In the jargon of derivatives, this is to say that for an action from F to become executable, what is left of E has to have become nullable (i.e., one can consider the execution of E completed). With reordering derivatives, we can execute an action from F successfully even when what is left of E is not nullable. It suffices that some sequence of actions to complete the residual of E is reorderable with the selected action of F.

In the definitions of the derivative operations we only use I in one direction, i.e., we do not make use of its symmetry. It would be interesting to see if our results can be generalized to the setting of semi-commutations [6] and which changes are required for that.

References

1 IJsbrand Jan Aalbersberg and Emo Welzl. Trace Languages Defined by Regular String Languages. *Theor. Inf. Appl.*, 20(2):103–119, 1986. doi:10.1051/ita/1986200201031.

2 Valentin M. Antimirov. Partial Derivatives of Regular Expressions and Finite Automaton Constructions. *Theor. Comput. Sci.*, 155(2):291–319, 1996. doi:10.1016/0304-3975(95)00182-4.

3 Alberto Bertoni, Giancarlo Mauri, and Nicoletta Sabadini. Unambiguous Regular Trace Languages. In Janos Demetrovics, Gyula Katona, and Arto Salomaa, editors, *Algebra, Combinatorics, and Logic in Computer Science*, volume 42 of *Colloquia Mathematica Societas János Bolyai*, pages 113–123. North-Holland, 1986.

4 Sabine Broda, António Machiavelo, Nelma Moreira, and Rogério Reis. Partial Derivative Automaton for Regular Expressions with Shuffle. In Jeffrey Shallit and Alexander Okhotin, editors, *Descriptional Complexity of Formal Systems: 17th International Workshop, DCFS 2015, Waterloo, ON, Canada, June 25-27, 2015, Proceedings*, volume 9118 of *Lecture Notes in Computer Science*, pages 21–32. Springer, 2015. doi:10.1007/978-3-319-19225-3_2.

5 Janusz A. Brzozowski. Derivatives of Regular Expressions. *J. ACM*, 11(4):481–494, 1964. doi:10.1145/321239.321249.

6 Mireille Clerbout and Michel Latteux. Semi-commutations. *Inf. Comput.*, 73(1):59–74, 1987. doi:10.1016/0890-5401(87)90040-X.

7 Kosaburo Hashiguchi. Recognizable Closures and Submonoids of Free Partially Commutative Monoids. *Theor. Comput. Sci.*, 86(2):233–241, 1991. doi:10.1016/0304-3975(91)90019-X.

8 Tony Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent Kleene Algebra and Its Foundations. *J. Log. Algebr. Program.*, 80(6):266–296, 2011. doi:10.1016/j.jlap.2011.04.005.

9 Stephen C. Kleene. Representation of Events in Nerve Sets and Finite Automata. In Claude E. Shannon and John McCarthy, editors, *Automata Studies*, volume 34 of *Annals of Mathematics Studies*, pages 3–42. Princeton University Press, 1956.

10 Barbara Klunder, Edward Ochmański, and Krystyna Stawikowska. On Star-Connected Flat Languages. *Fund. Inf.*, 67(1–3):93–105, 2005. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi67-1-3-08.

11 Dexter Kozen. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. *Inf. Comput.*, 110(2):366–390, 1994. doi:10.1006/inco.1994.1037.

12 Hendrik Maarand and Tarmo Uustalu. Reordering Derivatives of Trace Closures of Regular Languages. arXiv preprint 1908.03551, 2019. arXiv:1908.03551.

13 Antoni Mazurkiewicz. Concurrent Program Schemes and Their Interpretations. DAIMI Rep. PB-78, University of Aarhus, 1978.

14 Antoni Mazurkiewicz. Introduction to Trace Theory. In Volker Diekert, editor, *The Book of Traces*, pages 3–41. World Scientific, 1995. doi:10.1142/9789814261456_0001.

15 Edward Ochmański. Regular Behaviour of Concurrent Systems. *Bull. EATCS*, 27:56–67, 1985.

16 Edward Ochmański. Recognizable Trace Languages. In Volker Diekert, editor, *The Book of Traces*, pages 167–204. World Scientific, 1995. doi:10.1142/9789814261456_0006.

17 Michael O. Rabin and Dana S. Scott. Finite Automata and Their Decision Problems. *IBM J. Res. Devel.*, 3(2):114–125, 1959. doi:10.1147/rd.32.0114.

18 Jacques Sakarovitch. On Regular Trace Languages. *Theor. Comput. Sci.*, 52:59–75, 1987. doi:10.1016/0304-3975(87)90080-6.

19 Jacques Sakarovitch. The “Last” Decision Problem for Rational Trace Languages. 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 460–473. Springer, 1992. doi:10.1007/BFb0023848.

20 Martin Sulzmann and Peter Thiemann. Derivatives for Regular Shuffle Expressions. In Adrian-Horia Dediu, Enrico Formenti, Carlos Martín-Vide, and Bianca Truthe, editors, *Language and Automata Theory and Applications: 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*, volume 8977 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2015. doi:10.1007/978-3-319-15579-1_21.

21 Wiesław Zielonka. Notes on Finite Asynchronous Automata. *Theor. Inf. Appl.*, 21(2):99–135, 1987. doi:10.1051/ita/1987210200991.

22 Wiesław Zielonka. Asynchronous Automata. In Volker Diekert, editor, *The Book of Traces*, pages 205–247. World Scientific, 1995. doi:10.1142/9789814261456_0007.

Kleene Algebra with Observations

Tobias Kappé
University College London, UK
tkappe@cs.ucl.ac.uk

Paul Brunet
University College London, UK

Jurriaan Rot
University College London, UK
Radboud University, Nijmegen, The Netherlands

Alexandra Silva
University College London, UK

Jana Wagemaker
University College London, UK

Fabio Zanasi
University College London, UK

Abstract

Kleene algebra with tests (*KAT*) is an algebraic framework for reasoning about the control flow of sequential programs. Generalising KAT to reason about concurrent programs is not straightforward, because axioms native to KAT in conjunction with expected axioms for concurrency lead to an anomalous equation. In this paper, we propose *Kleene algebra with observations* (*KAO*), a variant of KAT, as an alternative foundation for extending KAT to a concurrent setting. We characterise the free model of KAO, and establish a decision procedure w.r.t. its equational theory.

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

Keywords and phrases Concurrent Kleene algebra, Kleene algebra with tests, free model, axiomatisation, decision procedure

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2019.41

Related Version A full version of the paper is available at `https://arxiv.org/abs/1811.10401`.

Funding ERC Starting Grant ProFoundNet (679127)
Jurriaan Rot: Marie Curie Fellowship (795119)
Alexandra Silva: Leverhulme Prize (PLP–2016–129)
Fabio Zanasi: EPSRC grant (EP/R020604/1)

Acknowledgements We are grateful to Jean-Baptiste Jeannin, Dan Frumin and Damien Pous individually, for their input which helped contextualise this work.

1 Introduction

The axioms of *Kleene algebra* (*KA*) [22, 9] correspond well to program composition [14], making them a valuable tool for studying equivalences between programs from an algebraic perspective. An extension of Kleene algebra known as *Kleene algebra with tests* (*KAT*) [24] adds primitives for conditional branching, and is particularly useful when proving validity of program transformations, such as optimisations applied by a compiler [28, 41].

As a matter of fact, KAT is sufficiently abstract to express not only program behaviour, but also program specifications; consequently, its laws can be used to compare programs to specifications [25, 1]. What makes this connection especially powerful is that KA (resp. KAT)

30th International Conference on Concurrency Theory (CONCUR 2019).
Editors: Wan Fokkink and Rob van Glabbeek; Article No. 41; pp. 41:1–41:16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is known to be *sound* and *complete* with respect to a language model [5, 30, 23, 29], meaning that an equation is valid in any KA (resp. KAT) precisely when it holds in the corresponding language model. Practical algorithms for deciding language equivalence [17, 6, 34] enable checking equations in KA or KAT, and hence automated verification becomes feasible [10].

More recently, Kleene algebra has been extended with a parallel composition operator, yielding *concurrent Kleene algebra* (*CKA*) [15, 13, 16]. Crucially, CKA includes the *exchange law*, which encodes *interleaving*, i.e., the (partial) sequentialisation of threads. Like its predecessors, CKA can be applied to verify (concurrent) programs by reasoning about equivalences [16]. The equational theory of CKA has also been characterised in terms of a language-like semantics [32, 21], where equivalence is known to be decidable [7].

Since both KAT and CKA are conservative extensions of KA, this prompted Jipsen and Moshier [19] to study a marriage between the two, dubbed *concurrent Kleene algebra with tests* (*CKAT*). The aim of CKAT is to extend CKA with Boolean guards, and thus arrive at a new algebraic perspective on verification of concurrent programs with conditional branching.

The starting point of this paper is the realisation that CKAT is not a suitable model of concurrent programs. This is because for any test p and CKAT-term e, one can prove $p \cdot e \cdot \overline{p} \equiv_{\mathsf{CKAT}} 0$, an equation that appears to have no reasonable interpretation for programs. The derivation goes as follows:

$$0 \leqq_{\mathsf{KAT}} p \cdot e \cdot \overline{p} \leqq_{\mathsf{CKA}} e \parallel (p \cdot \overline{p}) \equiv_{\mathsf{KAT}} e \parallel 0 \equiv_{\mathsf{CKA}} 0 \ .$$

As we shall see, this is possible because of the interplay between the exchange law and the fact that KAT identifies conjunction of tests with their sequential composition. For sequential programs, this identification is perfectly reasonable. In the context of concurrency with interleaving, however, actions from another thread may be scheduled in between two sequentially composed tests, whereas the conjunction of tests executes atomically. Indeed, an action scheduled between the tests might very well change the result of the second test.

It thus appears that, to reason algebraically about programs with both tests and concurrency, one needs a perspective on conditional branching where the conjunction of two tests is not necessarily the same as executing one test after the other. The remit of this paper is to propose an alternative to KAT, which we call *Kleene algebra with observations* (*KAO*), that makes exactly this distinction. We claim that, because of this change, KAO is more amenable to a sensible extension with primitives for concurrency. Establishing the meta-theory of KAO turns out to be a technically demanding task. We therefore devote this paper to such foundations, and leave development of concurrent KAO to follow-up work.

Concretely, we characterise the equational theory of KAO in terms of a language model (Section 5). Furthermore, we show that we can decide equality of these languages (and hence the equational theory of KAO) by deciding language equivalence of non-deterministic finite automata (Section 6). Both proofs show a clear separation of concerns: their kernel is idiomatic to KAO, and some well-known results from KA complete the argument.

For space reasons, detailed proofs are only included in the full version of this paper [20]; here, we sketch the main insights needed to prove the core propositions and theorems.

2 Preliminaries

We start by outlining some concepts and elementary results.

Boolean algebra. We use 2 to denote the set $\{0, 1\}$. The *powerset* (i.e., set of subsets) of a set S is denoted 2^S. We fix a finite set Ω of symbols called *observables*.

The *propositional terms* over Ω, denoted $\mathcal{T}_P$, are generated by the grammar

$$p, q ::= \bot \mid \top \mid o \in \Omega \mid p \vee q \mid p \wedge q \mid \overline{p} \;.$$

We write $\equiv_{\mathsf{BA}}$ for the smallest congruence on $\mathcal{T}_P$ that satisfies the axioms of Boolean algebra, i.e., such that for all $p, q, r \in \mathcal{T}_P$ the following hold:

$$p \vee \bot \equiv_{\mathsf{BA}} p \qquad p \vee q \equiv_{\mathsf{BA}} q \vee p \qquad p \vee \overline{p} \equiv_{\mathsf{BA}} \top \qquad p \vee (q \vee r) \equiv_{\mathsf{BA}} (p \vee q) \vee r$$

$$p \wedge \top \equiv_{\mathsf{BA}} p \qquad p \wedge q \equiv_{\mathsf{BA}} q \wedge p \qquad p \wedge \overline{p} \equiv_{\mathsf{BA}} \bot \qquad p \wedge (q \wedge r) \equiv_{\mathsf{BA}} (p \wedge q) \wedge r$$

$$p \vee (q \wedge r) \equiv_{\mathsf{BA}} (p \vee q) \wedge (p \vee r) \qquad p \wedge (q \vee r) \equiv_{\mathsf{BA}} (p \wedge q) \vee (p \wedge r) \;.$$

The set of *atoms*, denoted $\mathcal{A}$, is defined as 2^{Ω}. The semantics of propositional terms is given by the map $[\![-]\!]_{\mathsf{BA}} : \mathcal{T}_P \to 2^{\mathcal{A}}$, as follows:

$$[\![\bot]\!]_{\mathsf{BA}} = \emptyset \qquad [\![o]\!]_{\mathsf{BA}} = \{\alpha \in \mathcal{A} : o \in \alpha\} \qquad [\![p \vee q]\!]_{\mathsf{BA}} = [\![p]\!]_{\mathsf{BA}} \cup [\![q]\!]_{\mathsf{BA}}$$

$$[\![\top]\!]_{\mathsf{BA}} = \mathcal{A} \qquad [\![\overline{p}]\!]_{\mathsf{BA}} = \mathcal{A} \setminus [\![p]\!]_{\mathsf{BA}} \qquad [\![p \wedge q]\!]_{\mathsf{BA}} = [\![p]\!]_{\mathsf{BA}} \cap [\![q]\!]_{\mathsf{BA}} \;.$$

We also write $p \leqq_{\mathsf{BA}} q$ as a shorthand for $p \vee q \equiv_{\mathsf{BA}} q$.

It is known that $[\![-]\!]_{\mathsf{BA}}$ characterises $\equiv_{\mathsf{BA}}$ (c.f. [4, Chapter 5.9]), in the following sense:

▶ **Theorem 2.1** (Completeness for BA). *Let $p, q \in \mathcal{T}_P$; now $p \equiv_{\mathsf{BA}} q$ if and only if $[\![p]\!]_{\mathsf{BA}} = [\![q]\!]_{\mathsf{BA}}$.*

When $\alpha \in \mathcal{A}$, we write π_α for the Boolean term $\bigwedge_{o \in \alpha} o \wedge \bigwedge_{o \in \Omega \setminus \alpha} \overline{o}$, in which $\bigwedge$ is the obvious generalisation of $\wedge$ for some (arbitrary) choice of bracketing and order on terms. The following is then straightforward to prove.

▶ **Lemma 2.2.** *For all $\alpha \subseteq \Omega$ it holds that $[\![\pi_\alpha]\!]_{\mathsf{BA}} = \{\alpha\}$.*

Kleene algebra. A *word* over a set Δ is a sequence of symbols $d_0 \cdots d_{n-1}$ from Δ. The *empty word* is denoted ϵ. A set of words is called a *language*. Words can be *concatenated*: if w and x are words, then wx is the word where the symbols of w precede those of x. If L and L' are languages over Δ, then $L \cdot L'$ is the language of pairwise concatenations from L and L', i.e., $\{wx : w \in L, x \in L'\}$. We write $L^\star$ for the *Kleene closure* of L, which is the set $\{w_0 \cdots w_{n-1} : w_0, \dots, w_{n-1} \in L\}$. This makes $\Delta^\star$ the set of all words over Δ.

We fix a finite set of symbols Σ called the *alphabet*. The *rational terms* over Σ, denoted $\mathcal{T}_R$, are generated by the grammar

$$e, f ::= 0 \mid 1 \mid a \in \Sigma \mid e + f \mid e \cdot f \mid e^\star \;.$$

We write $\equiv_{\mathsf{KA}}$ for the smallest congruence on $\mathcal{T}_R$ that satisfies the axioms of Kleene algebra, i.e., such that for all $e, f, g \in \mathcal{T}_R$ the following hold:

$$e + 0 \equiv_{\mathsf{KA}} e \qquad e + e \equiv_{\mathsf{KA}} e \qquad e + f \equiv_{\mathsf{KA}} f + e \qquad e + (f + g) \equiv_{\mathsf{KA}} (e + f) + g$$

$$e \cdot 1 \equiv_{\mathsf{KA}} e \qquad e \equiv 1 \cdot e \qquad e \cdot 0 \equiv_{\mathsf{KA}} 0 \qquad 0 \equiv 0 \cdot e \qquad e \cdot (f \cdot g) \equiv_{\mathsf{KA}} (e \cdot f) \cdot g$$

$$e \cdot (f + g) \equiv_{\mathsf{KA}} e \cdot f + e \cdot g \qquad 1 + e \cdot e^\star \equiv_{\mathsf{KA}} e^\star \qquad e + f \cdot g \leqq_{\mathsf{KA}} g \implies f^\star \cdot e \leqq_{\mathsf{KA}} g$$

$$(e + f) \cdot g \equiv_{\mathsf{KA}} e \cdot g + f \cdot g \qquad 1 + e^\star \cdot e \equiv_{\mathsf{KA}} e^\star \qquad e + f \cdot g \leqq_{\mathsf{KA}} f \implies e \cdot g^\star \leqq_{\mathsf{KA}} f \;,$$

in which $e \leqq_{\mathsf{KA}} f$ is a shorthand for $e + f \equiv_{\mathsf{KA}} f$.

The semantics of rational terms is given by $[\![-]\!]_{\mathsf{KA}} : \mathcal{T}_R \to 2^{\Sigma^\star}$, in the following sense:

$$[\![0]\!]_{\mathsf{KA}} = \emptyset \qquad [\![a]\!]_{\mathsf{KA}} = \{a\} \qquad [\![e+f]\!]_{\mathsf{KA}} = [\![e]\!]_{\mathsf{KA}} \cup [\![f]\!]_{\mathsf{KA}}$$
$$[\![1]\!]_{\mathsf{KA}} = \{\epsilon\} \qquad [\![e^\star]\!]_{\mathsf{KA}} = [\![e]\!]_{\mathsf{KA}}^\star \qquad [\![e \cdot f]\!]_{\mathsf{KA}} = [\![e]\!]_{\mathsf{KA}} \cdot [\![f]\!]_{\mathsf{KA}} \ .$$

It is furthermore known that $[\![-]\!]_{\mathsf{KA}}$ characterises $\equiv_{\mathsf{KA}}$ [5, 30, 23], as follows:

▶ **Theorem 2.3** (Completeness for KA). *Let $e, f \in \mathcal{T}_R$; now $e \equiv_{\mathsf{KA}} f$ if and only if $[\![e]\!]_{\mathsf{KA}} = [\![f]\!]_{\mathsf{KA}}$.*

We also work with matrices and vectors of rational terms. Let Q be a finite set. A *Q-vector* is a function $x : Q \to \mathcal{T}_R$; a *Q-matrix* is a function $M : Q \times Q \to \mathcal{T}_R$.

Let $e \in \mathcal{T}_R$, let x and y be Q-vectors, and let M be a Q-matrix. *Addition* of vectors is defined pointwise, i.e., $x + y$ is the Q-vector given by $(x + y)(q) = x(q) + y(q)$. We can also *scale* vectors, writing $x \fatsemi e$ for the Q-vector given by $(x \fatsemi e)(q) = x(q) \cdot e$.

Multiplication of a Q-vector by a Q-matrix yields a Q-vector, as expected:

$$(M \cdot x)(q) = \sum_{q' \in Q} M(q, q') \cdot x(q') \ .$$

Here, $\sum$ is the usual generalisation of $+$ for some (arbitrary) choice of bracketing and order on terms; the empty sum is defined to be 0.

We write $x \equiv_{\mathsf{KA}} y$ when x and y are pointwise equivalent, i.e., for all $q \in Q$ we have $x(q) \equiv_{\mathsf{KA}} y(q)$; we extend $\leqq_{\mathsf{KA}}$ to Q-vectors as before. Matrices over rational terms again obey the axioms of Kleene algebra [23]. As a special case, we can obtain the following:

▶ **Lemma 2.4.** *Let M be a Q-matrix. Using the entries of M and applying the operators of Kleene algebra, we can construct a Q-matrix $M^\star$, which has the following property. Let y be any Q-vector; now $M^\star \cdot y$ is the least (w.r.t. $\leqq_{\mathsf{KA}}$) Q-vector x such that $M \cdot x + y \leqq_{\mathsf{KA}} x$.*

Automata and bisimulations. We briefly recall *bisimulation up to congruence* for language equivalence of automata, from [6]. This will be used in Section 6.

A *non-deterministic automaton* (*NDA*) over an alphabet Σ is a triple (X, o, d) where $o \colon X \to 2$ is an output function, and $d \colon X \times \Sigma \to X$ a transition function. A non-deterministic finite automaton (NFA) is an NDA where X is finite. It will be convenient to characterise the semantics of an NDA (X, o, d) recursively as the unique map $\ell : X \to 2^{\Sigma^\star}$ such that

$$\ell(x) = \{\epsilon : o(x) = 1\} \cup \bigcup_{x' \in d(x,a)} \{a\} \cdot \ell(x') \ .$$

This coincides with the standard definition of language acceptance for NDAs. The *determinisation* of an NDA (X, o, d) is the deterministic automaton $(2^X, \overline{o}, \overline{d})$ [35], where

$$\overline{o}(V) = \begin{cases} 1 & \text{if } \exists s \in V \text{ s.t. } o(s) = 1 \\ 0 & \text{otherwise} \end{cases} \qquad \overline{d}(V, a) = \bigcup_{s \in V} d(s, a) \ .$$

The *congruence closure* of a relation $R \subseteq 2^X \times 2^X$, denoted R^c, is the least equivalence relation such that $R \subseteq R^c$, and if $(C, D) \in R^c$ and $(E, F) \in R^c$ then $(C \cup E, D \cup F) \in R^c$. A *bisimulation up to congruence* for an NDA (X, o, d) is a relation $R \subseteq 2^X \times 2^X$ such that for all $(V, W) \in R$, we have $\overline{o}(V) = \overline{o}(W)$ and furthermore for all $a \in \Sigma$, we have $(\overline{d}(V, a), \overline{d}(W, a)) \in R^c$. Bisimulations up to congruence give a proof technique for language equivalence of states of non-deterministic automata, as a consequence of the following [6].

▶ **Theorem 2.5.** *Let (X, o, d) be an NDA. For all $U, V \in 2^X$, we have $\bigcup_{x \in U} \ell(x) = \bigcup_{x \in V} \ell(x)$ if and only if there exists a bisimulation up to congruence R for (X, o, d) such that $(U, V) \in R$.*

In [6], it is shown how the construction of bisimulations up to congruence leads to a very efficient algorithm for language equivalence.

3 The problem with CKAT

Our motivation for adjusting the axioms of KAT is based on the fact that conjunction and sequential composition are distinct when interleaving is involved, because sequential composition leaves a "gap" between tests that might be used by another thread, possibly changing the outcome of the second test. We now formalise this, by detailing how combining KAT and CKA to obtain CKAT (as in [19]) leads to the absurd equation $p \cdot e \cdot \overline{p} \equiv_{\mathsf{CKAT}} 0$.

Kleene algebra with tests. To obtain KAT, we enrich rational terms with propositions; concretely, the set of *guarded rational terms* over Σ and Ω, denoted $\mathcal{T}_{GR}$, is generated by

$$e, f ::= 0 \mid 1 \mid a \in \Sigma \mid p \in \mathcal{T}_P \mid e + f \mid e \cdot f \mid e^\star .$$

We define $\equiv_{\mathsf{KAT}}$ as the smallest congruence on $\mathcal{T}_{GR}$ which contains $\equiv_{\mathsf{BA}}$ and obeys the axioms of $\equiv_{\mathsf{KA}}$ (e.g., $e + e \equiv_{\mathsf{KAT}} e$). Furthermore, $\equiv_{\mathsf{KAT}}$ should relate constants and operators on propositional subterms: for all $p, q \in \mathcal{T}_P$ it holds that[1]

$$\bot \equiv_{\mathsf{KAT}} 0 \qquad \top \equiv_{\mathsf{KAT}} 1 \qquad p \vee q \equiv_{\mathsf{KAT}} p + q \qquad p \wedge q \equiv_{\mathsf{KAT}} p \cdot q .$$

Guarded rational terms relate to programs by viewing actions as statements and propositions as assertions. The last axiom is therefore not strange: if we assert $\mathtt{x} = \mathtt{1}$ followed by $\mathtt{y} = \mathtt{1}$ then surely, if $\mathtt{x}$ and $\mathtt{y}$ remain the same, this is equivalent to asserting $\mathtt{x} = \mathtt{1} \wedge \mathtt{y} = \mathtt{1}$.

Concurrent Kleene algebra. To obtain CKA, we add a parallel composition operator to KA. Concretely, the set of *series-rational terms* [33] over Σ, denoted $\mathcal{T}_{SR}$, is generated by

$$e, f ::= 0 \mid 1 \mid a \in \Sigma \mid e + f \mid e \cdot f \mid e \parallel f \mid e^\star .$$

We define $\equiv_{\mathsf{CKA}}$ as the smallest congruence on $\mathcal{T}_{SR}$ which obeys the axioms that generate $\equiv_{\mathsf{KA}}$, as well as the following for all $e, f, g, h \in \mathcal{T}_{SR}$:

$$e \parallel f \equiv_{\mathsf{CKA}} f \parallel e \qquad e \parallel 1 \equiv_{\mathsf{CKA}} e \qquad e \parallel 0 \equiv_{\mathsf{CKA}} 0 \qquad (e + f) \parallel g \equiv_{\mathsf{CKA}} e \parallel g + f \parallel g$$

$$e \parallel (f \parallel g) \equiv_{\mathsf{CKA}} (e \parallel f) \parallel g \qquad (e \parallel f) \cdot (g \parallel h) \leqq_{\mathsf{CKA}} (e \cdot g) \parallel (f \cdot h) ,$$

in which $e \leqq_{\mathsf{CKA}} f$ is an abbreviation for $e + f \equiv_{\mathsf{CKA}} f$.

Here, 0 annihilates parallel composition because 0 corresponds to the program without valid traces; hence, running e in parallel with 0 also cannot yield any traces. Distributivity of $\parallel$ over $+$ witnesses that a choice within a thread can also be made outside that thread.

The last axiom, called the *exchange law* [15], encodes interleaving. Intuitively, it says that when programs $e \cdot g$ and $f \cdot h$ run in parallel, their behaviour includes running their heads in parallel (i.e., $e \parallel f$) followed by their tails (i.e., $g \parallel h$). Taken to its extreme, the exchange law says that the behaviour of a program includes its complete linearisations.

[1] This is a slightly contrived definition of KAT; one usually presents the constants and operators using the same symbols [24]. To contrast KAO and KAT it is helpful to make this identification explicit.

Concurrent Kleene algebra with tests. To define CKAT [19], we choose *guarded series-rational terms* over Σ and Ω, denoted $\mathcal{T}_{GSR}$, as those generated by the grammar

$$e, f ::= 0 \mid 1 \mid a \in \Sigma \mid p \in \mathcal{T}_P \mid e + f \mid e \cdot f \mid e \parallel f \mid e^\star \ .$$

Now, $\equiv_{\mathsf{CKAT}}$ is the least congruence on $\mathcal{T}_{GSR}$ obeying the axioms of $\equiv_{\mathsf{KAT}}$ and $\equiv_{\mathsf{CKA}}$.

If we view guarded series-rational terms as representations of programs, and use $\equiv_{\mathsf{CKAT}}$ to reason about them, then we should expect to obtain sensible statements about programs, since the axioms that underpin CKAT seem reasonable in that context.

To test this hypothesis, consider the guarded series-rational term $p \cdot e \cdot \overline{p}$, which represents a program that first performs an assertion p, then runs a program described by e, and asserts the negation of p. There are choices of p and e that should make $p \cdot e \cdot \overline{p}$ describe a program with valid behaviour; for instance, p could assert that $\mathtt{x} = \mathtt{1}$, and e could be the program $\mathtt{x} \leftarrow \mathtt{0}$. Hence, by our hypothesis, $p \cdot e \cdot \overline{p}$ should not, in general, be equivalent to 0, the program without any valid behaviour. Unfortunately, the opposite is true.

▶ **Antinomy 3.1.** *Let $e \in \mathcal{T}_{GSR}$ and $p \in \mathcal{T}_P$; now $p \cdot e \cdot \overline{p} \equiv_{\mathsf{CKAT}} 0$.*

Proof. By the axiom that 1 is the unit of $\parallel$ and the exchange law, we derive that

$$p \cdot e \cdot \overline{p} \equiv_{\mathsf{CKAT}} (p \parallel 1) \cdot (1 \parallel e) \cdot \overline{p} \leqq_{\mathsf{CKAT}} ((p \cdot 1) \parallel (1 \cdot e)) \cdot \overline{p} \ .$$

By the same reasoning, and the axiom that 1 is the unit of $\cdot$, we have that

$$((p \cdot 1) \parallel (1 \cdot e)) \cdot \overline{p} \equiv_{\mathsf{CKAT}} (p \parallel e) \cdot (\overline{p} \parallel 1) \leqq_{\mathsf{CKAT}} (p \cdot \overline{p}) \parallel (e \cdot 1) \ .$$

Applying the unit, and using the identification of $\wedge$ and $\cdot$ on tests, we find that

$$(p \cdot \overline{p}) \parallel (e \cdot 1) \equiv_{\mathsf{CKAT}} (p \cdot \overline{p}) \parallel e \equiv_{\mathsf{CKAT}} (p \wedge \overline{p}) \parallel e \equiv_{\mathsf{CKAT}} \bot \parallel e \ .$$

Since $\bot$ and 0 are identified, and 0 annihilates parallel composition, we have that

$$\bot \parallel e \equiv_{\mathsf{CKAT}} 0 \parallel e \equiv_{\mathsf{CKAT}} 0 \ .$$

By the above, we have $p \cdot e \cdot \overline{p} \leqq_{\mathsf{CKAT}} 0$. Since $0 \leqq_{\mathsf{CKAT}} p \cdot e \cdot \overline{p}$, the claim follows. ◀

Another way of contextualising the above is to consider that propositional Hoare logic can be encoded in KAT [26]. Concretely, a propositional Hoare triple $\{p\}S\{q\}$ is valid if $p \cdot e_S \cdot \overline{q} \equiv_{\mathsf{KAT}} 0$, where e_S is a straightforward encoding of S. It stands to reason that sequential programs should similarly encode in CKAT. However, if we apply that line of reasoning to the above, then *any* Hoare triple of the form $\{p\}S\{p\}$ is valid, which would mean that *any property is an invariant of any program.*

4 Kleene algebra with observations

We now propose KAO as an alternative way of embedding Boolean guards in rational terms. This approach should prevent Antinomy 3.1, and thus make KAO more suitable for an extension with concurrency. We start by motivating how we adapt KAT, before proceeding with a language model and a generalisation of partial derivatives to KAO.

4.1 From tests to observations

The root of the problem in Antinomy 3.1 is the axiom $p \cdot q \equiv_{\mathsf{KAT}} p \wedge q$, telling us that $(p \cdot \overline{p}) \parallel e \equiv_{\mathsf{CKAT}} (p \wedge \overline{p}) \parallel e$. Interpreted as programs, these are different: in one, e can be interleaved between p and $\overline{p}$, which the other does not allow.

To fix this problem, we propose a new perspective on Boolean guards. Rather than considering a guard a *test*, which in KAT entails an assertion valid from the last action to the next, we consider a guard an *observation*, i.e., an assertion valid at that particular point in the execution of the program. This weakens the connection between conjunction and concatenation: if a program observes p followed by q, there is no guarantee that p and q can be observed simultaneously. Hence, we drop the equivalence between conjunction and concatenation of tests. We call this system *weak Kleene algebra with observations*; like KAT, its axioms are based on the axioms of Boolean algebra and Kleene algebra, as follows.

▶ **Definition 4.1** (WKAO axioms). *We define $\equiv_{\mathsf{WKAO}}$ as the smallest congruence on $\mathcal{T}_{GR}$ that contains $\equiv_{\mathsf{BA}}$ and also obeys the axioms of $\equiv_{\mathsf{KA}}$. Furthermore, $\bot \equiv_{\mathsf{WKAO}} 0$ and for all $p, q \in \mathcal{T}_P$ it holds that $p \vee q \equiv_{\mathsf{WKAO}} p + q$.*

Since conjunction and concatenation no longer coincide, we have also dropped the axiom that relates their units. This can be justified from our shift in perspective: $\top$, i.e., the observation that always succeeds, is an action that leaves some record in the behaviour of the program, whereas 1, i.e., the program that does nothing, has no such obligation.[2]

As hinted before, it may happen that when a program observes p followed by q, both these assertions are true simultaneously, i.e., their conjunction could have been observed; hence, the behaviour of observing p *and* q should be contained in the behaviour of observing p *and then* q. For instance, consider the pseudo-programs S = `assert` $p \wedge q$; e and S' = `assert` p; `assert` q; e. Here, S asserts that p and q hold at the same time before proceeding with e, while S' first asserts p, and then q. The behaviour of S should be included in that of S': if p and q are simultaneously observable, then the first two observations might take place simultaneously. Encoding this in an additional axiom leads to KAO proper.

▶ **Definition 4.2** (KAO axioms). *We define $\equiv_{\mathsf{KAO}}$ as the smallest congruence on $\mathcal{T}_{GR}$ that contains $\equiv_{\mathsf{WKAO}}$, and furthermore satisfies the* contraction law*: for all $p, q \in \mathcal{T}_P$ it holds that $p \wedge q \leqq_{\mathsf{KAO}} p \cdot q$ – where $e \leqq_{\mathsf{KAO}} f$ is a shorthand for $e + f \equiv_{\mathsf{KAO}} f$.*

We briefly return to our example programs. If we encode S as $(p \wedge q) \cdot e$ and S' as $p \cdot q \cdot e$, then the behaviour of S is contained in that of S', because $(p \wedge q) \cdot e \leqq_{\mathsf{KAO}} p \cdot q \cdot e$.

▶ Remark 4.3. In the presence of the other axioms, $p \wedge q \leqq_{\mathsf{KAO}} p \cdot q$ is equivalent to $p \leqq_{\mathsf{KAO}} p \cdot p$. Indeed, the second inequality may be inferred from the first, since $p \equiv_{\mathsf{KAO}} p \wedge p$. The converse implication is obtained as follows: $p \wedge q \leqq_{\mathsf{KAO}} (p \wedge q) \cdot (p \wedge q) \leqq_{\mathsf{KAO}} p \cdot q$, using that $p \wedge q \leqq_{\mathsf{KAO}} p, q$. While the contraction law is more convenient to compare terms, the axiom $p \leqq_{\mathsf{KAO}} p \cdot p$ will serve implicitly as the basis for the contraction relation $\preceq$ defined in the next section.

4.2 A language model

The Kleene algebra variants encountered thus far have models based on rational languages, which characterise the equivalences derivable using the axioms. To find a model for guarded rational terms which corresponds to KAO, we start with a model of weak KAO:

[2] We refer to [20, Appendix E] for further algebraic details on the consequences of identifying these units.

▶ **Definition 4.4** (WKAO semantics). *Let* $\Gamma = \mathcal{A} \cup \Sigma$. *Languages over* Γ *are called* observation languages. *We define* $[\![-]\!]_{\mathsf{WKAO}} : \mathcal{T}_{GR} \to 2^{\Gamma^\star}$ *as follows:*

$$[\![0]\!]_{\mathsf{WKAO}} = \emptyset \qquad [\![a]\!]_{\mathsf{WKAO}} = \{a\} \qquad [\![e+f]\!]_{\mathsf{WKAO}} = [\![e]\!]_{\mathsf{WKAO}} \cup [\![f]\!]_{\mathsf{WKAO}} \qquad [\![e^\star]\!]_{\mathsf{WKAO}} = [\![e]\!]^\star_{\mathsf{WKAO}}$$
$$[\![1]\!]_{\mathsf{WKAO}} = \{\epsilon\} \qquad [\![p]\!]_{\mathsf{WKAO}} = [\![p]\!]_{\mathsf{BA}} \qquad [\![e \cdot f]\!]_{\mathsf{WKAO}} = [\![e]\!]_{\mathsf{WKAO}} \cdot [\![f]\!]_{\mathsf{WKAO}} \ ,$$

where it is understood that $[\![p]\!]_{\mathsf{BA}} \subseteq \mathcal{A} \subseteq \Gamma \subseteq \Gamma^\star$.

Observation languages are a model for guarded rational terms w.r.t. $\equiv_{\mathsf{WKAO}}$:

▶ **Lemma 4.5** (WKAO soundness). *Let* $e, f \in \mathcal{T}_{GR}$. *Now, if* $e \equiv_{\mathsf{WKAO}} f$, *then* $[\![e]\!]_{\mathsf{WKAO}} = [\![f]\!]_{\mathsf{WKAO}}$.

More work is necessary to get a model of guarded rational terms w.r.t. $\equiv_{\mathsf{KAO}}$. In particular, $[\![-]\!]_{\mathsf{WKAO}}$ does not preserve the contraction law: if $o_1, o_2 \in \Omega$, then

$$[\![o_1 \wedge o_2]\!]_{\mathsf{WKAO}} = \{\alpha \in \mathcal{A} : o_1, o_2 \in \alpha\} \qquad [\![o_1 \cdot o_2]\!]_{\mathsf{WKAO}} = \{\alpha\beta \in \mathcal{A} \cdot \mathcal{A} : o_1 \in \alpha, o_2 \in \beta\} \ ,$$

meaning that $[\![o_1 \wedge o_2]\!]_{\mathsf{WKAO}} \not\subseteq [\![o_1 \cdot o_2]\!]_{\mathsf{WKAO}}$, despite $o_1 \wedge o_2 \leqq_{\mathsf{KAO}} o_1 \cdot o_2$.[3]

To obtain a sound model, we need the counterpart of the contraction law on the level of observation languages, which works out as follows

▶ **Definition 4.6** (Contraction). *We define* $\preceq$ *as the smallest relation on* $\Gamma^\star$ *s.t.*

$$\frac{w, x \in \Gamma^\star \qquad \alpha \in \mathcal{A}}{w\alpha x \preceq w\alpha\alpha x} \ .$$

When $L \subseteq \Gamma^\star$, *we write* $L{\downarrow}$ *for the* $\preceq$*-closure of* L, *that is to say, the smallest observation language such that* $L \subseteq L{\downarrow}$, *and if* $w \preceq x$ *with* $x \in L{\downarrow}$, *then* $w \in L{\downarrow}$.

Thus, $L{\downarrow}$ contains all words obtained from words in L by contracting any number of repeated atoms (but not actions) into one. Applying this closure to the semantics encodes the intuition that repeated observations may also correspond to just one step in the execution.

With these tools, we can define a model of KAO as follows.

▶ **Definition 4.7** (KAO semantics). *We define* $[\![-]\!]_{\mathsf{KAO}} : \mathcal{T}_{GR} \to 2^{\Gamma^\star}$ *by* $[\![e]\!]_{\mathsf{KAO}} = [\![e]\!]_{\mathsf{WKAO}}{\downarrow}$.

Alternatively, we can describe $[\![-]\!]_{\mathsf{KAO}}$ compositionally, using the following.

▶ **Lemma 4.8.** *Let* $e, f \in \mathcal{T}_{GR}$. *The following hold: (i)* $[\![e+f]\!]_{\mathsf{KAO}} = [\![e]\!]_{\mathsf{KAO}} \cup [\![f]\!]_{\mathsf{KAO}}$, *and (ii)* $[\![e \cdot f]\!]_{\mathsf{KAO}} = ([\![e]\!]_{\mathsf{KAO}} \cdot [\![f]\!]_{\mathsf{KAO}}){\downarrow}$, *and (iii)* $[\![e^\star]\!]_{\mathsf{KAO}} = [\![e]\!]^\star_{\mathsf{KAO}}{\downarrow}$.

It is now straightforward to check that $[\![-]\!]_{\mathsf{KAO}}$ preserves the axioms of $\equiv_{\mathsf{KAO}}$.

▶ **Lemma 4.9** (KAO soundness). *Let* $e, f \in \mathcal{T}_{GR}$. *Now, if* $e \equiv_{\mathsf{KAO}} f$, *then* $[\![e]\!]_{\mathsf{KAO}} = [\![f]\!]_{\mathsf{KAO}}$.

4.3 Partial derivatives

Derivatives [8] are a powerful tool in Kleene algebra variants. In the context of programs, we can think of derivatives as an operational semantics; they tell us whether the term represents a program that can halt immediately (given by the *termination map*), as well as the program that remains to be executed once an action is performed (given by the *continuation map*).

[3] Incidentally, this shows that the contraction law is independent of the axioms that build $\equiv_{\mathsf{WKAO}}$, i.e., that those axioms are not sufficient to prove the contraction law.

Derivatives are typically compatible with the congruences used to reason about terms; what's more, they are closely connected to finite automata [8]. This makes them useful for reasoning about language models [26, 6, 34]. We therefore introduce a form of derivatives of guarded rational terms that works well with $\equiv_{\mathsf{KAO}}$. Let us start by giving the termination map, which is close to the termination map of rational terms compatible with $\equiv_{\mathsf{KA}}$.

▶ **Definition 4.10** (Termination). *We define $\epsilon : \mathcal{T}_{GR} \to 2$ inductively, as follows:*

$$\epsilon(0) = 0 \qquad \epsilon(a) = 0 \qquad \epsilon(e + f) = \max\{\epsilon(e), \epsilon(f)\} \qquad \epsilon(e^\star) = 1$$
$$\epsilon(1) = 1 \qquad \epsilon(p) = 0 \qquad \epsilon(e \cdot f) = \min\{\epsilon(e), \epsilon(f)\} \ .$$

To check that ϵ characterises guarded rational terms that can terminate, we record the following lemma, which says that $\epsilon(e) = 1$ precisely when $[\![e]\!]_{\mathsf{KAO}}$ includes the empty string.

▶ **Lemma 4.11.** *Let $e \in \mathcal{T}_{GR}$. Now $\epsilon(e) \leqq_{\mathsf{KAO}} e$, and $\epsilon(e) = 1$ if and only if $\epsilon \in [\![e]\!]_{\mathsf{KAO}}$.*

Next we define the continuation map, or more specifically, *partial continuation map* [2]: given $\mathfrak{a} \in \Gamma$, this function gives a *set* of terms representing possible continuations of the program after performing $\mathfrak{a}$. We start with the continuation map for actions.

▶ **Definition 4.12** (Σ-continuation). *We define $\delta : \mathcal{T}_{GR} \times \Sigma \to 2^{\mathcal{T}_{GR}}$ inductively*

$$\begin{aligned} \delta(0, a) = \delta(1, a) &= \emptyset & \delta(e + f, a) &= \delta(e, a) \cup \delta(f, a) \\ \delta(a, a') &= \{1 : a = a'\} & \delta(e \cdot f, a) &= \{e' \cdot f : e' \in \delta(e, a)\} \cup \Delta(e, f, a) \\ \delta(p, a) &= \emptyset & \delta(e^\star, a) &= \{e' \cdot e^\star : e' \in \delta(e, a)\} \ , \end{aligned}$$

where $\Delta(e, f, a) = \delta(f, a)$ when $\epsilon(e) = 1$, and $\emptyset$ otherwise.

Finally, we give the continuation map for observations, which is similar to the one for actions, except that on sequential composition it is subtly different.

▶ **Definition 4.13** ($\mathcal{A}$-continuation). *We define $\zeta : \mathcal{T}_{GR} \times \mathcal{A} \to 2^{\mathcal{T}_{GR}}$ inductively*

$$\begin{aligned} \zeta(0, \alpha) = \delta(1, \alpha) &= \emptyset & \zeta(e + f, \alpha) &= \zeta(e, \alpha) \cup \zeta(f, \alpha) \\ \zeta(a, \alpha) &= \emptyset & \zeta(e \cdot f, \alpha) &= \{e' \cdot f : e' \in \zeta(e, \alpha)\} \cup Z(e, f, \alpha) \\ \zeta(p, \alpha) &= \{1 : \pi_\alpha \leqq_{\mathsf{BA}} p\} & \zeta(e^\star, \alpha) &= \{e' \cdot e^\star : e' \in \zeta(e, \alpha)\} \ , \end{aligned}$$

where $Z(e, f, \alpha) = \zeta(f, \alpha)$ when $\epsilon(e) = 1$ or $\epsilon(e') = 1$ for some $e' \in \zeta(e, \alpha)$, and $\emptyset$ otherwise.

For instance, let $p, q \in \mathcal{T}_P$. If $\alpha \in [\![p]\!]_{\mathsf{BA}} \cap [\![q]\!]_{\mathsf{BA}}$, then we can calculate that

$$\zeta(p \cdot q, \alpha) = \{p' \cdot q : p' \in \zeta(p, \alpha)\} \cup Z(p, q, \alpha) = \{1 \cdot q\} \cup \{1\} \ ,$$

which is to say that the program can either continue in $1 \cdot q$, where it has to make an observation validating q, or it can choose to re-use the observation α to validate q, continuing in $Z(p, q, \alpha) = \zeta(q, \alpha) = \{1\}$, because $1 \in \zeta(p, \alpha)$. This notion that ζ can apply an observation more than once to validate multiple assertions in a row can be formalised.

▶ **Lemma 4.14.** *Let $e \in \mathcal{T}_{GR}$, $\alpha \in \mathcal{A}$, and $e' \in \zeta(e, \alpha)$. Now $\zeta(e', \alpha) \subseteq \zeta(e, \alpha)$.*

To check that δ and ζ indeed output continuations of the term after the action or assertion has been performed, we should check that they are compatible with $\equiv_{\mathsf{KAO}}$.

▶ **Lemma 4.15.** *Let $e \in \mathcal{T}_{GR}$. For all $a \in \Sigma$ and $e' \in \delta(e, a)$, we have $a \cdot e' \leqq_{\mathsf{KAO}} e$. Furthermore, for all $\alpha \in \mathcal{A}$ and $e' \in \zeta(e, \alpha)$, we have $\pi_\alpha \cdot e' \leqq_{\mathsf{KAO}} e$.*

We also need the *reach* of a guarded rational term, which is meant to describe the set of terms that can be obtained by repeatedly applying continuation maps.

▶ **Definition 4.16** (Reach of a term). *For $e \in \mathcal{T}_{GR}$, we define $\rho : \mathcal{T}_{GR} \to 2^{\mathcal{T}_{GR}}$ inductively:*

$$\rho(0) = \emptyset \qquad \rho(a) = \{1, a\} \qquad \rho(e + f) = \rho(e) \cup \rho(f)$$
$$\rho(1) = \{1\} \qquad \rho(p) = \{1, p\} \qquad \rho(e \cdot f) = \{e' \cdot f : e' \in \rho(e)\} \cup \rho(f)$$
$$\rho(e^\star) = \{1\} \cup \{e' \cdot e^\star : e' \in \rho(e)\} \ .$$

Indeed, $\rho(e)$ truly contains all terms reachable from e by means of δ and ζ:

▶ **Lemma 4.17.** *Let $e \in \mathcal{T}_{GR}$. If $a \in \Sigma$, then $\delta(e, a) \subseteq \rho(e)$; if $e' \in \rho(e)$, then $\delta(e', a) \subseteq \rho(e)$. Furthermore, if $\alpha \in \mathcal{A}$, then $\zeta(e, \alpha) \subseteq \rho(e)$; if $e' \in \rho(e)$, then $\zeta(e', \alpha) \subseteq \rho(e)$.*

It is not hard to see that for all $e \in \mathcal{T}_{GR}$, we have that $\rho(e)$ is finite. Note that $\rho(e)$ does not, in general, contain e itself. On the other hand, $\rho(e)$ does contain a set of terms that is sufficient to reconstruct e, up to $\equiv_{\mathsf{KAO}}$. We describe these as follows.

▶ **Definition 4.18** (Initial factors). *For $e \in \mathcal{T}_{GR}$, we define $\iota : \mathcal{T}_{GR} \to 2^{\mathcal{T}_{GR}}$ inductively:*

$$\iota(0) = \emptyset \qquad \iota(a) = \{a\} \qquad \iota(e + f) = \iota(e) \cup \iota(f)$$
$$\iota(1) = \{1\} \qquad \iota(p) = \{p\} \qquad \iota(e \cdot f) = \{e' \cdot f : e' \in \iota(e)\}$$
$$\iota(e^\star) = \{1\} \cup \{e' \cdot e^\star : e' \in \iota(e)\} \ .$$

Now, to reconstruct e from $\iota(e)$, all we have to do is sum its elements.

▶ **Lemma 4.19.** *Let $e \in \mathcal{T}_{GR}$. Then $e \equiv_{\mathsf{KAO}} \sum_{e' \in \iota(e)} e'$.*

5 Completeness

We now set out to show that $[\![-]\!]_{\mathsf{KAO}}$ characterises $\equiv_{\mathsf{KAO}}$. Since soundness of $[\![-]\!]_{\mathsf{KAO}}$ w.r.t. $\equiv_{\mathsf{KAO}}$ was already shown in Lemma 4.9, it remains to prove completeness, i.e., if e and f are interpreted equally by $[\![-]\!]_{\mathsf{KAO}}$, then they can be proven equivalent via $\equiv_{\mathsf{KAO}}$. To this end, we first identify a subset of guarded rational terms for which a completeness result can be shown by relying on the completeness result for KA. To describe these terms, we need the following.

▶ **Definition 5.1** (Atomic and guarded terms). *Let Π denote the set $\{\pi_\alpha : \alpha \in \mathcal{A}\}$. The set of* atomic guarded rational terms*, denoted $\mathcal{T}_{AGR}$, is the subset of $\mathcal{T}_{GR}$ generated by the grammar*

$$e, f ::= 0 \ | \ 1 \ | \ a \in \Sigma \ | \ \pi_\alpha \in \Pi \ | \ e + f \ | \ e \cdot f \ | \ e^\star \ .$$

Furthermore, if $e \in \mathcal{T}_{GR}$ such that $[\![e]\!]_{\mathsf{KAO}} = [\![e]\!]_{\mathsf{WKAO}}$, we say that e is closed.

Completeness of $\equiv_{\mathsf{KAO}}$ with respect to $[\![-]\!]_{\mathsf{KAO}}$ can then be established for atomic and closed guarded rational terms as follows.

▶ **Proposition 5.2.** *Let $e, f \in \mathcal{T}_{AGR}$ be closed, and $[\![e]\!]_{\mathsf{KAO}} = [\![f]\!]_{\mathsf{KAO}}$; then $e \equiv_{\mathsf{KAO}} f$.*

Sketch. By noting that the semantics of an atomic and closed term coincide with the KA-semantics (over an extended alphabet); the result then follows by appealing to completeness of KA, and the fact that the axioms of KA are contained in those for KAO. ◀

To show that $[\![-]\!]_{\mathsf{KAO}}$ characterises $\equiv_{\mathsf{KAO}}$ for *all* guarded rational terms, it suffices to find for every $e \in \mathcal{T}_{GR}$ a closed $\hat{e} \in \mathcal{T}_{AGR}$ with $e \equiv_{\mathsf{KAO}} \hat{e}$. After all, if we have such a transformation, then $[\![e]\!]_{\mathsf{KAO}} = [\![f]\!]_{\mathsf{KAO}}$ implies $[\![\hat{e}]\!]_{\mathsf{KAO}} = [\![\hat{f}]\!]_{\mathsf{KAO}}$, which implies $\hat{e} \equiv_{\mathsf{KAO}} \hat{f}$, and hence $e \equiv_{\mathsf{KAO}} f$.

In the remainder of this section, we describe how to obtain a closed and atomic guarded rational term from a guarded rational term e. This transformation works by first "disassembling" e using partial derivatives; on an intuitive level, this is akin to creating a (non-deterministic finite) automaton that accepts the observation language described by e. Next, we "reassemble" from this representation an atomic term $\hat{e}$, equivalent to e; this is analogous to constructing a rational expression from the automaton. To show that $\hat{e}$ is closed, we leverage Lemma 4.14, essentially arguing that the automaton obtained from e encodes $\preceq$.

We extend the notation for vectors and matrices over rational terms to guarded rational terms. We can then represent a guarded rational term by a matrix and a vector, as follows.

▶ **Definition 5.3.** *For $e \in \mathcal{T}_{GR}$, define the $\rho(e)$-vector x_e and $\rho(e)$-matrix M_e by*

$$x_e(e') = \epsilon(e') \qquad\qquad M_e(e', e'') = \sum_{e'' \in \delta(e', a)} a + \sum_{e'' \in \zeta(e', \alpha)} \pi_\alpha \ .$$

Note that, in the above, $\delta(e', a)$ and $\zeta(e', \alpha)$ are finite by Lemma 4.17.

Kozen's argument that matrices over rational terms satisfy the axioms of Kleene algebra [23] generalises to guarded rational terms. This leads to a straightforward generalisation of Lemma 2.4. For the sake of brevity, we use T to stand in for WKAO or KAO.

▶ **Lemma 5.4.** *Let M be a Q-matrix. Using the entries of M and applying the operators of Kleene algebra, we can construct a Q-matrix $M^\star$, which has the following property. Let y be any Q-vector; now $M^\star \cdot y$ is the least (w.r.t. $\leqq_{\mathsf{T}}$) Q-vector x such that $M \cdot x + y \leqq_{\mathsf{T}} x$.*

We can now use the above to reassemble a term from M_e and x_e as follows.

▶ **Definition 5.5** (Transformation of terms). *Let $e \in \mathcal{T}_{GR}$. We write s_e for the $\rho(e)$-vector given by $M_e^\star \cdot x_e$, and $\hat{e}$ for the guarded rational term given by $\sum_{e' \in \iota(e)} s_e(e')$.*

Since $\hat{e}$ is constructed from M_e and s_e, and these are built using atomic guarded rational terms, $\hat{e}$ is atomic. We carry on to show that $\hat{e} \equiv_{\mathsf{KAO}} e$. To this end, the following is useful.

▶ **Proposition 5.6** (Least solutions). *Let $e, f \in \mathcal{T}_{GR}$. Now $s_e \fatsemi f$ is the least (w.r.t. $\leqq_{\mathsf{T}}$) $\rho(e)$-vector s such that for each $e' \in \rho(e)$, it holds that*

$$\epsilon(e') \cdot f + \sum_{e'' \in \delta(e', a)} a \cdot s(e'') + \sum_{e'' \in \zeta(e', \alpha)} \pi_\alpha \cdot s(e'') \leqq_{\mathsf{T}} s(e') \ .$$

When $s = s_e \fatsemi f$, the above is an equivalence, i.e., we can substitute $\leqq_{\mathsf{T}}$ with $\equiv_{\mathsf{T}}$.

Sketch. The least such s coincides with $s_e = M_e^\star \cdot x_e$, by using the property of s_e that we obtain as a result of Lemma 5.4. The latter claim goes by standard argument akin to the argument showing that the least pre-fixpoint of a monotone operator is a fixpoint. ◀

Using the above, we show that the contents of $\rho(e)$ themselves qualify as a least $\rho(e)$-vector satisfying system obtained from e (and fixing $f = 1$). More concretely, we have the following.

▶ **Proposition 5.7.** *Let $e \in \mathcal{T}_{GR}$ and $e' \in \rho(e)$. Then $s_e(e') \equiv_{\mathsf{KAO}} e'$.*

Sketch. The key idea is to first relate the least $\rho(e)$-vectors satisfying the system obtained from e to those satisfying the systems arising from its subterms. The proof then follows by induction on e, and some straightforward derivations. ◀

This allows us to conclude (using Lemma 4.19) that $e \equiv_{\mathsf{KAO}} \hat{e}$.

▶ **Corollary 5.8** (Transformation preserves equivalence). *Let $e \in \mathcal{T}_{GR}$; then $e \equiv_{\mathsf{KAO}} \hat{e}$.*

It remains to show that $\hat{e}$ is closed. To this end, the following characterisation is helpful:

▶ **Lemma 5.9.** *Let $L \subseteq \Gamma^\star$, and define $\trianglelefteq$ as the smallest relation on $\Gamma^\star$ satisfying the rules*

$$\frac{}{\epsilon \trianglelefteq \epsilon} \qquad \frac{\mathfrak{a} \in \Gamma \qquad w \trianglelefteq x}{\mathfrak{a}w \trianglelefteq \mathfrak{a}x} \qquad \frac{\alpha \in \mathcal{A} \qquad w \trianglelefteq x}{\alpha w \trianglelefteq \alpha\alpha x} .$$

$L{\downarrow}$ is the smallest subset of $\Gamma^\star$ such that $L \subseteq L{\downarrow}$, and if $w \trianglelefteq x$ with $x \in L{\downarrow}$, then $w \in L{\downarrow}$.

We can then employ the characterisation of s_e in Proposition 5.6 to show, by induction on the length of the $\trianglelefteq$-chain, that the components of s_e are closed.

▶ **Proposition 5.10.** *Let $e \in \mathcal{T}$. For all $e' \in \rho(e)$, it holds that $s_e(e')$ is closed.*

Sketch. Note that s_e satisfies the system obtained from e as in Proposition 5.6, both w.r.t. $\equiv_{\mathsf{KAO}}$ and $\equiv_{\mathsf{WKAO}}$. Using Lemma 5.9, we can then show (by induction on the number of applications of $\trianglelefteq$) that $[\![s(e')]\!]_{\mathsf{WKAO}} \subseteq [\![s(e')]\!]_{\mathsf{KAO}}$; the other inclusion holds trivially. ◀

Hence, we can conclude (using Lemma 4.8) that $\hat{e}$ is closed as well, as follows.

▶ **Corollary 5.11** (Transformation yields closed term). *Let $e \in \mathcal{T}_{GR}$; now $\hat{e}$ is closed.*

We are now ready conclude with the main result of this section.

▶ **Theorem 5.12** (Soundness and completeness). *Let $e, f \in \mathcal{T}_{GR}$; then*

$$e \equiv_{\mathsf{KAO}} f \iff [\![e]\!]_{\mathsf{KAO}} = [\![f]\!]_{\mathsf{KAO}} .$$

Sketch. Since we have a way to obtain from $e \in \mathcal{T}_{GR}$ a closed term $\hat{e} \in \mathcal{T}_{AGR}$, for which it furthermore holds that $e \equiv_{\mathsf{KAO}} \hat{e}$, we can appeal to Proposition 5.2. ◀

6 Decision procedure

We now design a procedure that takes terms $e, f \in \mathcal{T}_{GR}$ and decides whether $[\![e]\!]_{\mathsf{KAO}} = [\![f]\!]_{\mathsf{KAO}}$; by Theorem 5.12, this gives us a decision procedure for $e \equiv_{\mathsf{KAO}} f$. To this end, we reduce to the problem of language equivalence between NFAs over guarded rational terms, defined using partial derivatives. This, in turn, can be efficiently decided using bisimulation techniques.

First, observe that $\mathcal{T}_{GR}$ carries a non-deterministic automaton structure.

▶ **Definition 6.1** (Syntactic automaton). *The set $\mathcal{T}_{GR}$ carries an NDA structure $(\mathcal{T}_{GR}, \epsilon, \theta)$ where $\epsilon : \mathcal{T}_{GR} \to 2$ is as in Definition 4.10, and $\theta : \mathcal{T}_{GR} \times \Gamma \to 2^{\mathcal{T}_{GR}}$ is given by*

$$\theta(e, \mathfrak{a}) = \begin{cases} \delta(e, \mathfrak{a}) & \text{if } \mathfrak{a} \in \Sigma \\ \zeta(e, \mathfrak{a}) & \text{if } \mathfrak{a} \in \mathcal{A} \end{cases} .$$

We call this the syntactic automaton *of guarded rational terms.*

Let $\ell \colon \mathcal{T}_{GR} \to 2^{\Gamma^\star}$ be the semantics of this automaton as given in Section 2. It is easy to see that ℓ is the unique function such that

$$\ell(e) = \{\epsilon : \epsilon(e) = 1\} \cup \bigcup_{e' \in \delta(e,a)} \{a\} \cdot \ell(e') \cup \bigcup_{e' \in \zeta(e,\alpha)} \{\alpha\} \cdot \ell(e') . \tag{1}$$

To use this NDA for KAO equivalence, we need to show that the language accepted by the state $e \in \mathcal{T}_{GR}$ is $[\![e]\!]_{\mathsf{KAO}}$. For this goal, it helps to algebraically characterise expressions in terms of their derivatives. We call this a *fundamental theorem* of KAO, after [39, 40].

▶ **Theorem 6.2** (Fundamental theorem). *For all $e \in \mathcal{T}_{GR}$, the following holds*

$$e \equiv_{\mathsf{KAO}} \epsilon(e) + \sum_{e' \in \delta(e,a)} a \cdot e' + \sum_{e' \in \zeta(e,\alpha)} \pi_\alpha \cdot e' \ .$$

Sketch. The right-to-left containment is a result of Lemmas 4.11 and 4.15. The converse is a straightforward calculation using Proposition 5.6 and Corollary 5.8. ◀

Next, we turn the fundamental theorem into a statement about the KAO semantics. Before we do, however, we need the following basic result about the KAO semantics.

▶ **Lemma 6.3.** *Let us fix $\alpha \in \mathcal{A}$. For all $e \in \mathcal{T}_{GR}$, we have:*

$$\left[\!\!\left[\sum\nolimits_{e' \in \zeta(e,\alpha)} \pi_\alpha \cdot e'\right]\!\!\right]_{\mathsf{KAO}} = \bigcup_{e' \in \zeta(e,\alpha)} \{\alpha\} \cdot [\![e']\!]_{\mathsf{KAO}} \ .$$

We now arrive at a semantic analogue of the fundamental theorem.

▶ **Proposition 6.4.** *For all $e \in \mathcal{T}_{GR}$, we have:*

$$[\![e]\!]_{\mathsf{KAO}} = \{\epsilon : \epsilon(e) = 1\} \cup \bigcup_{e' \in \delta(e,a)} \{a\} \cdot [\![e']\!]_{\mathsf{KAO}} \cup \bigcup_{e' \in \zeta(e,\alpha)} \{\alpha\} \cdot [\![e']\!]_{\mathsf{KAO}} \ .$$

Sketch. By applying soundness of $\equiv_{\mathsf{KAO}}$ w.r.t. $[\![-]\!]_{\mathsf{KAO}}$ (Lemma 4.9) to the fundamental theorem, and using Lemma 6.3 for the term summing over the $\mathcal{A}$-continuations of e. ◀

The language of a state in the syntactic NFA is then characterised by the KAO semantics.

▶ **Theorem 6.5** (Soundness of translation). *Let $e \in \mathcal{T}_{GR}$; then $[\![e]\!]_{\mathsf{KAO}} = \ell(e)$.*

Sketch. A direct consequence of Proposition 6.4 and the uniqueness of ℓ in satisfying (1). ◀

To decide whether $[\![e]\!]_{\mathsf{KAO}} = [\![f]\!]_{\mathsf{KAO}}$ it suffices to show that e and f are language equivalent in the syntactic automaton. We express this in terms of ι as defined in Definition 4.18.

▶ **Corollary 6.6.** *For all $e, f \in \mathcal{T}_{GR}$, we have*

$$[\![e]\!]_{\mathsf{KAO}} = [\![f]\!]_{\mathsf{KAO}} \quad \Leftrightarrow \quad \bigcup_{e' \in \iota(e)} \ell(e') = \bigcup_{f' \in \iota(f)} \ell(f') \ .$$

By construction, $\iota(e) \subseteq \rho(e)$ for every $e \in \mathcal{T}_{GR}$. Since $\rho(e)$ and $\rho(f)$ are closed under partial derivatives, we can restrict the syntactic automaton to $\rho(e) \cup \rho(f)$, to obtain a *finite* NDA. To decide $e \equiv_{\mathsf{KAO}} f$, it suffices to decide $\bigcup_{e' \in \iota(e)} \ell(e') = \bigcup_{f' \in \iota(f)} \ell(f')$ on this NFA.

This leads us to the main result of this section: a decision procedure for KAO.

▶ **Theorem 6.7** (Decision procedure). *For all $e, f \in \mathcal{T}_{GR}$, we have $e \equiv_{\mathsf{KAO}} f$ if and only if there exists R such that $(\iota(e), \iota(f)) \in R$ and R is a bisimulation up to congruence for the syntactic automaton restricted to $\rho(e) \cup \rho(f)$.*

Sketch. By Theorem 2.5, such an R exists precisely when $\bigcup_{e' \in \iota(e)} [\![e']\!]_{\mathsf{KAO}} = \bigcup_{f' \in \iota(f)} [\![f']\!]_{\mathsf{KAO}}$; by Corollary 6.6 and Theorem 5.12 this is equivalent to $e \equiv_{\mathsf{KAO}} f$. ◀

▶ **Example 6.8.** We know that for all $a, b \in \Omega$ we have that $a \wedge b \not\equiv_{\mathsf{KAO}} a \cdot b$. This also follows from our decision procedure, which we argue by showing that any attempt to construct a bisimulation up to congruence R containing $(\{a \wedge b\}, \{a \cdot b\})$ fails.

We start with $R = \{(\{a \wedge b\}, \{a \cdot b\})\}$, and note that $\bar{\epsilon}(\{a \wedge b\}) = 0 = \bar{\epsilon}(\{a \cdot b\})$. We now take a derivative w.r.t. $\alpha \in \mathcal{A}$ such that $\pi_\alpha \leqq_{\mathsf{BA}} a \wedge b$. To grow R into a bisimulation up to congruence, all derivatives should be checked and possibly added to R; this specific choice, however, will lead to a counterexample. We have that $\bar{\theta}(\{a \wedge b\}, \alpha) = \{1\}$ and $\bar{\theta}(\{a \cdot b\}, \alpha) = \{1 \cdot b, 1\}$. We add $(\{1\}, \{1 \cdot b, 1\})$ to R, noting that $\bar{\epsilon}(\{1\}) = 1 = \bar{\epsilon}(\{1 \cdot b, 1\})$, and continue with the next derivative w.r.t. $\beta \in \mathcal{A}$ such that $\pi_\beta \leqq_{\mathsf{BA}} b$. We get $\bar{\theta}(\{1\}, \beta) = \emptyset$ and $\bar{\theta}(\{1 \cdot b, 1\}, \beta) = \{1\}$. We now have $(\emptyset, \{1\}) \in R$, but $\bar{\epsilon}(\emptyset) \neq \bar{\epsilon}(\{1\})$. Thus, we cannot construct a bisimulation up to congruence R such that $(\{a \wedge b\}, \{a \cdot b\}) \in R$.

▶ Remark 6.9. For KAO-expressions without observations, the derivatives with respect to an element of $\mathcal{A}$ result in the empty set. Hence, for these KAO-expressions, deciding equivalence comes down to standard derivative-based techniques for rational expressions [37].

7 Related work

This work fits in the larger tradition of Kleene algebra as a presentation of the "laws of programming", the latter having been studied by Hoare and collaborators [14, 13]. More precisely, our efforts can be grouped with recent efforts to extend Kleene algebra with concurrency [15, 13, 32, 21], and thence with Boolean guards [19].

We proved that $\equiv_{\mathsf{KAO}}$ is sound and complete w.r.t. $[\![-]\!]_{\mathsf{KAO}}$, based on the existing completeness proof for KA. By re-using completeness results for a simpler algebra, the proof shows a clear separation of concerns between the "old" algebra being extended and the new layer of axioms placed on top. This strategy pops up quite often in some form [29, 1, 32, 21]. We note that unlike [29, 1], our transformation does not proceed by induction on the term, but leverages the least fixpoints computable in every Kleene algebra. Further, the combination of KA with additional hypotheses presented in [27] might yield another route to completeness.

The use of linear systems to study automata was pioneered by Conway [9] and Backhouse [3]. Kozen's completeness proof expanded on this by generalising Kleene algebra to matrices [23]. The connection between linear systems, derivatives and completeness was studied by Kozen [26] and Jacobs [18]. To keep our presentation simple, we give our proof of completeness in elementary terms; a proof in terms of coalgebra [38] may yet be possible.

Using bisimulation to decide language equivalence in a deterministic finite automaton originates from Hopcroft and Karp [17]. This technique has many generalisations [36].

8 Conclusions and further work

Kleene algebra with observations (KAO) is an algebraic framework that adds Boolean guards to Kleene algebra in such a way that a sensible extension with concurrency is still possible, in contrast with Kleene algebra with tests (KAT). Indeed, the laws of KAO prevent the problem presented by Antinomy 3.1. The axiomatisation of KAO, as well as the decision procedure for equivalence, give an alternative foundation for combining KAO with concurrent Kleene algebra to arrive at a new equational calculus of concurrent programs.

The most obvious direction of further work is to define *concurrent Kleene algebra with observations* (*CKAO*) as the amalgamation of axioms of KAO and CKA, in pursuit of a characterisation of its equational theory and an analogous decidability result. We conjecture that languages of *sp-pomsets* [12, 11, 33] over actions and atoms, closed under some suitable relation, form the free model; a proof would likely build on [31] and re-use techniques from [21]. While the equational theory of CKAO might be decidable, we are less optimistic about a feasible algorithm; deciding the equational theory of CKA is already EXPSPACE-complete [7].

Another avenue of future research would be to create a programming language using KAO (or, possibly, CKAO) by instantiating actions and observations, and adding axioms that encode the intention of those primitives. NetKAT [1, 10, 41], a language for describing and reasoning about network behaviour is an excellent example of such an endeavour based on KAT. Our long-term hope is that CKAO will function as a foundation for a "concurrent" version of NetKAT aimed at describing and reasoning about networks with concurrency.

Finally, we note that the decision procedure for KAO based on bisimulation up to congruence leaves room for optimisation. Besides adapting the work on symbolic algorithms for bisimulation-based algorithms in KAT [34], the transitivity property witnessed in Lemma 4.14 seems like it could sometimes allow a bisimulation-based algorithm to decide early.

References

1 Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. NetKAT: semantic foundations for networks. In *POPL*, pages 113–126, 2014. doi:10.1145/2535838.2535862.

2 Valentin M. Antimirov. Partial Derivatives of Regular Expressions and Finite Automaton Constructions. *Theor. Comput. Sci.*, 155(2):291–319, 1996. doi:10.1016/0304-3975(95)00182-4.

3 Roland Backhouse. *Closure algorithms and the star-height problem of regular languages.* PhD thesis, University of London, 1975.

4 Garrett Birkhoff and Thomas C. Bartee. *Modern applied algebra.* McGraw-Hill, 1970.

5 Maurice Boffa. Une remarque sur les systèmes complets d'identités rationnelles. *ITA*, 24:419–428, 1990.

6 Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In *POPL*, pages 457–468, 2013. doi:10.1145/2429069.2429124.

7 Paul Brunet, Damien Pous, and Georg Struth. On Decidability of Concurrent Kleene Algebra. In *CONCUR*, pages 28:1–28:15, 2017. doi:10.4230/LIPIcs.CONCUR.2017.28.

8 Janusz A. Brzozowski. Derivatives of Regular Expressions. *J. ACM*, 11(4):481–494, 1964. doi:10.1145/321239.321249.

9 John Horton Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, 1971.

10 Nate Foster, Dexter Kozen, Matthew Milano, Alexandra Silva, and Laure Thompson. A Coalgebraic Decision Procedure for NetKAT. In *POPL*, pages 343–355, 2015. doi:10.1145/2676726.2677011.

11 Jay L. Gischer. The Equational Theory of Pomsets. *Theor. Comput. Sci.*, 61:199–224, 1988. doi:10.1016/0304-3975(88)90124-7.

12 Jan Grabowski. On partial languages. *Fundam. Inform.*, 4(2):427, 1981.

13 Tony Hoare. Laws of Programming: The Algebraic Unification of Theories of Concurrency. In *CONCUR*, pages 1–6, 2014. doi:10.1007/978-3-662-44584-6_1.

14 Tony Hoare, Ian J. Hayes, Jifeng He, Carroll Morgan, A. W. Roscoe, Jeff W. Sanders, Ib Holm Sørensen, J. Michael Spivey, and Bernard Sufrin. Laws of Programming. *Commun. ACM*, 30(8):672–686, 1987. doi:10.1145/27651.27653.

15 Tony Hoare, Bernhard Möller, Georg Struth, and Ian Wehrman. Concurrent Kleene Algebra. In *CONCUR*, pages 399–414, 2009. doi:10.1007/978-3-642-04081-8_27.

16 Tony Hoare, Stephan van Staden, Bernhard Möller, Georg Struth, and Huibiao Zhu. Developments in Concurrent Kleene Algebra. *J. Log. Algebr. Meth. Program.*, 85(4):617–636, 2016. doi:10.1016/j.jlamp.2015.09.012.

17 John E. Hopcroft and Richard M. Karp. A linear algorithm for testing equivalence of finite automata. Technical Report TR71-114, Cornell University, December 1971.

18 Bart Jacobs. A Bialgebraic Review of Deterministic Automata, Regular Expressions and Languages. In *Algebra, Meaning, and Computation (Joseph A. Goguen festschrift)*, pages 375–404, 2006. doi:10.1007/11780274_20.

19 Peter Jipsen and M. Andrew Moshier. Concurrent Kleene algebra with tests and branching automata. *J. Log. Algebr. Meth. Program.*, 85(4):637–652, 2016. doi:10.1016/j.jlamp.2015.12.005.

20 Tobias Kappé, Paul Brunet, Jurriaan Rot, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. Kleene Algebra with Observations. *CoRR*, abs/1811.10401, 2018. arXiv:1811.10401.

21 Tobias Kappé, Paul Brunet, Alexandra Silva, and Fabio Zanasi. Concurrent Kleene Algebra: Free Model and Completeness. In *ESOP*, pages 856–882, 2018. doi:10.1007/978-3-319-89884-1_30.

22 Stephen C. Kleene. Representation of Events in Nerve Nets and Finite Automata. *Automata Studies*, pages 3–41, 1956.

23 Dexter Kozen. A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events. *Inf. Comput.*, 110(2):366–390, 1994. doi:10.1006/inco.1994.1037.

24 Dexter Kozen. Kleene algebra with tests and commutativity conditions. In *TACAS*, pages 14–33, 1996. doi:10.1007/3-540-61042-1_35.

25 Dexter Kozen. On Hoare logic and Kleene algebra with tests. *ACM Trans. Comput. Log.*, 1(1):60–76, 2000. doi:10.1145/343369.343378.

26 Dexter Kozen. Myhill-Nerode Relations on Automatic Systems and the Completeness of Kleene Algebra. In *STACS*, pages 27–38, 2001. doi:10.1007/3-540-44693-1_3.

27 Dexter Kozen and Konstantinos Mamouras. Kleene Algebra with Equations. In *ICALP*, pages 280–292, 2014. doi:10.1007/978-3-662-43951-7_24.

28 Dexter Kozen and Maria-Christina Patron. Certification of Compiler Optimizations Using Kleene Algebra with Tests. In *CL*, pages 568–582, 2000. doi:10.1007/3-540-44957-4_38.

29 Dexter Kozen and Frederick Smith. Kleene Algebra with Tests: Completeness and Decidability. In *CSL*, pages 244–259, 1996. doi:10.1007/3-540-63172-0_43.

30 Daniel Krob. Complete Systems of B-Rational Identities. *Theor. Comput. Sci.*, 89(2):207–343, 1991. doi:10.1016/0304-3975(91)90395-I.

31 Michael R. Laurence and Georg Struth. Completeness Theorems for Bi-Kleene Algebras and Series-Parallel Rational Pomset Languages. In *RAMiCS*, pages 65–82, 2014. doi:10.1007/978-3-319-06251-8_5.

32 Michael R. Laurence and Georg Struth. Completeness Theorems for Pomset Languages and Concurrent Kleene Algebras, 2017. arXiv:1705.05896.

33 Kamal Lodaya and Pascal Weil. Series-parallel languages and the bounded-width property. *Theor. Comput. Sci.*, 237(1):347–380, 2000. doi:10.1016/S0304-3975(00)00031-1.

34 Damien Pous. Symbolic Algorithms for Language Equivalence and Kleene Algebra with Tests. In *POPL*, pages 357–368, 2015. doi:10.1145/2676726.2677007.

35 Michael O. Rabin and Dana S. Scott. Finite Automata and Their Decision Problems. *IBM J. of Research and Dev.*, 3(2):114–125, 1959. doi:10.1147/rd.32.0114.

36 Jurriaan Rot, Filippo Bonchi, Marcello M. Bonsangue, Damien Pous, Jan Rutten, and Alexandra Silva. Enhanced coalgebraic bisimulation. *Mathematical Structures in Comput. Sci.*, 27(7):1236–1264, 2017. doi:10.1017/S0960129515000523.

37 Jan J. M. M. Rutten. Automata and Coinduction (An Exercise in Coalgebra). In *CONCUR*, pages 194–218, 1998. doi:10.1007/BFb0055624.

38 Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.

39 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.

40 Alexandra Silva. *Kleene Coalgebra.* PhD thesis, Radboud Universiteit Nijmegen, 2010.

41 Steffen Smolka, Spiridon Aristides Eliopoulos, Nate Foster, and Arjun Guha. A fast compiler for NetKAT. In *ICFP*, pages 328–341, 2015. doi:10.1145/2784731.2784761.